```
            printf("\nFAILURE TO OPEN OR LOCATE FILE!!!\n");
            found_file = NO;
        }
        else
            found_file = YES;
    }


    printf("\n\nPlease enter the sequence number of the primary sequence:");
    scanf("%d",&mainseq);                                   /*query for mainseq parameter*/

    printf("\n\nEnter the INTEGER PRECISION of the P-values you will calculate(ex:4):");
    scanf("%d",&prec);                                      /*query for PRECISION parameter*/
    /* value of 42 -> "fast forwards" through scoring step */


    /* see score_manager function for use of these values below */
    printf("\n\nEnter the COLUMN POSITION start value (0-Total Positions):");
    scanf("%d", &START);

    printf("\n\nEnter the COLUMN POSITION STOP value (0-Total Positions; -1 = END):");
    scanf("%d", &STOP);


/* added to "jump" over score caculation step */

    if (prec == 42)
        PRECISION = 42;

    else
        PRECISION = (double) pow(10,prec);

/*********************************************/



    input = read_input(inputfile);

    fclose(inputfile);

    return input;

}



/*************************************************/
/********FUNCTION: read_input *********************/
/** returns pointer to input *********************/
/*************************************************/


char    *read_input(FILE *ifile)
{
    unsigned char *cvector(long nl, long nh);

    void exitprogram();

    extern int      NUM_ROW;
    extern float    NUM_COL;
    extern int      OFFSET;

    char    *input;

    int x;          /*int read*/
    char    c;          /*char read*/
    float   fmax_col;   /*max num of residues, float*/
    int imax_col;       /*max num of residues, int*/
    int seq = 0;            /*number of sequences = NUM_ROW*/

    int count = 0;      /*total number of aa + label characters*/
    int i = 0;      /*input index*/

    int reading = 1;    /*flag for EOF*/
```

```
    long temp;

    NUM_COL = 0;
    NUM_ROW = 0;
    OFFSET = 0;


    printf("\nAllocating Memory for Input File.\n");

    x = fgetc(ifile);
    while (x != EOF) {
        if (x == '>')
            NUM_ROW += 1;
        if (isalpha(x) != 0 || x == '-')
            count += 1;
        x = fgetc(ifile);
    }


    fmax_col = (float) (count / NUM_ROW);
    imax_col = (int) fmax_col + 1;

    temp = NUM_ROW*imax_col;
    input = cvector(0,temp);
    /*allocate input matrix */
    /*NOTE: matrix will be larger than needed*/


    if (input == NULL) {
        printf("\nDynamic Memory Allocation FAILURE!!\n");
        exitprogram();
        exit(1);
    }
    else
        printf("Allocation Completed.\n");


    fseek(ifile, 0, SEEK_SET);  /*start reading from begining of file*/

    printf("Attempting to Read File. Please Wait.\n");
    x = fgetc(ifile);
    while (reading == YES)  {
        while (x != '>')
            x = fgetc(ifile);
        while (x != '\n')
            x = fgetc(ifile);
        while ( x != '>' && x != EOF )    {
            if (isalpha(x) != 0 || x == '-')    {
                c = x;
                *(input + i) = c;
                i += 1;
            }
            x = fgetc(ifile);
        }
        if (x == EOF)
            reading = NO;
        printf("Sequence #%d Read. Continuing . . .\n", seq);
        seq += 1;
    }
    printf("\n\nFile Read.\n");

    if (i % NUM_ROW != 0)     {
        printf("ERROR: SEQEUNCES NOT EQUAL IN LENGTH:\n");
        printf("CHECK FOR PROPER ALIGNMENT\n");
        exitprogram();
        exit(5);
    }
    else     {
        NUM_COL = (float) i / NUM_ROW;
        OFFSET = (int) NUM_COL;
    }
    return input;
}
```

```
/************************************************/
/********FUNCTION: score_manager ****************/
/************************************************/


void    score_manager(char *input)
{

    int **chi_analysis(char *input, int start, int stop);

    extern float NUM_COL;
    extern int START;
    extern int STOP;

    int **chi_matrix;

    int start;
    int stop;


    start = START;

    if (STOP < 0)
        stop = (int) NUM_COL;              /*so that program can be run simutaneous on di
f. comp*/

    else
        stop = STOP;



    chi_matrix = chi_analysis(input, start, stop);

    free_imatrix(chi_matrix, 0, 30, 0, 30);

}




/************************************************/
/********FUNCTION: chi_analysis ******************/
/************************************************/

int **chi_analysis(char *input, int start, int stop)
{

    int     find_unique_elements(char *input, char *aa, int *gap, int var);
    void    crosstab(char *input, int **chi_matrix, char *aa1, char *aa2, int var1, int va
r2, int count1, int count2);
    int     chisquare(int **nn, int ni, int nj, float *chisq, float *df, double *prob, dou
ble *cramrv, float *ccc);
    void    gap_rectifer(int **chi_matrix,int gap1, int gap2,int *cnt1, int *cnt2);
    void    matrixconverter(int **chi_matrix, int *matrix, int count1, int count2, int fla
g);
    double  estExact(int *ROWmatrix, double PRECISION, int numrows, int numcols);
    int     cochrantest(int **chi_matrix, int *rowtot, int *coltot, float *expctd, int num
rows, int numcols);
    float   GetFloatTimer (float timer);


    extern float    NUM_COL;            /*Number of Residue Positions*/
    extern double   PRECISION;          /*precision of calculated probability*/
    extern int      NUM_ROW;            /*Number of Seqeunces*/
    extern int      CELLS;
    extern int      PLACES;
    extern int      *dlength;


    char    uni1[POSITIONS];                    /*array of unique aa in a position*/
    char    uni2[POSITIONS];                    /*array of unique aa in a 2nd pos */
    char    *unique1;
```

```
    char    *unique2;
    int count1;                         /*number of unique aa in a position*/
    int count2;                         /*number of unique aa in a 2nd pos */
    int i, j;
    int gap1, gap2;                 /*number location of gap (-) in uni arrays*/
    int original_count1;
    int test_flag;

    /** int iqw,jqw,kqw; ***/               /*loop variables for debugging display*/

    int *rowtot;                    /*array for row totals*/
    int *coltot;                    /*array for column totals*/
    float *expctd;                  /*matrix for expected frequncy table*/


    float   chi, d, cc;
    double pr,cram;
    float   *chisq, *df, *ccc;
    double  *prob, *cramrv;

    double estprob;                 /*probability as determined by exact or estimated exac
t methods*/

    double logprob;

    int **chi_matrix;                   /*CROSSTABULATION matrix of correlation frequency
*/
    int **chi_matx2;                    /*cross tabulation matrix with origin=ORIGIN */
                                        /*This ORIGIN shift is necessary for chisquare fun
ction*/

    float timer;

    int *matrix_by_rows;            /* 2D chi_matrix represented as 1D array, by rows */

    char    outfile[30] = "alldata";        /*output file for data*/
    char    fullname[40];                       /*output file for data*/

    char    outfile2[30] = "nulldata";      /*output file for meaningless data*/
    char    fullname2[40];

    char    outfile3[30] = "distribution.txt";  /*output for distribution data*/

    int cramfreq[1001] = {0};
    int index, index2, ww, tt;
    int *lpfreq;


    FILE * output = NULL;                       /*output file initiation*/
    FILE * output2 = NULL;
    FILE * output3 = NULL;


    /**** MEMORY ISSUES ****/


    unique1 = uni1;
    unique2 = uni2;
    chisq = &chi;
    df = &d;
    prob = &pr;
    cramrv = &cram;
    ccc = &cc;

    chi_matrix = imatrix(0, POSITIONS, 0, POSITIONS);     /*allocates memory for an int ma
trix */


    lpfreq = ivector(0,100*prec);   /* allocates memory for log-prob distribution */


    for (tt=0; tt <= 100*prec; tt++) {  /*initializes log-prob variable to 0 */
        lpfreq[tt] = 0;
    }
```

```
    /*****************************************/
    /*****************************************/
    /*         Open Files for output         */
    /*****************************************/
    /*****************************************/


    sprintf(fullname, "%s%dx%d.txt", outfile, start, stop);
    sprintf(fullname2, "%s%dx%d.txt", outfile2, start, stop);


    output = fopen(fullname, "w");
    output2 = fopen(fullname2, "w");
    output3 = fopen(outfile3, "w");


    if (output == NULL)      {
        printf("\nFailure to Create alldata File.\n");
        exit(2);
    }

    if (output2 == NULL)     {
        printf("\nFailure to Create nulldata File.\n");
        exit(2);
    }

    if (output3 == NULL)     {
        printf("\nFailure to Create DISTRIBUTION File.\n");
        exit(2);
    }


    /*****************************************/
    /* Start Timer */
    timer = GetFloatTimer (0.);
    /*****************************************/



    /*****************************************/
    /***end of opening file for output**********/
    /*****************************************/



    /*****************************************/
    /*****************************************/
    /**********  Start of Main Cycle ***********/
    /*****************************************/
    /*****************************************/


    for (i = start; i < stop; i++)   {

        printf("\n SCORING position #%d's possible interactions. . . please wait.", i);

        count1 = find_unique_elements(input, unique1, &gap1, i);                    /*find
s unique aa in 1st residue*/
        original_count1 = count1;


        for (j = (i+1); j < NUM_COL; j++)  {


            printf("\n\t\t SCORING its interaction with position #%d . . . calculating . .
 .", j);

            count1 = original_count1;           /*must be reset each cycle so gap_rectifie
r does not keep reducing number*/
            count2 = find_unique_elements(input, unique2, &gap2, j);                /*find
s unqiue aa in 2nd residue*/

        /*********************/
        /*form crosstab matrix*/
        /*********************/
```

```
            crosstab(input, chi_matrix, unique1, unique2, i, j, count1, count2);     /*make
s correlation table */
            gap_rectifer(chi_matrix,gap1, gap2, &count1, &count2);                   /*adju
sts for presence of gaps*/


        /*****************************************************************/
        /* Allocation of memory for reading by various analyitical programs */
        /*****************************************************************/

            rowtot=ivector(0,count1);                                        /*row totals*/
            coltot=ivector(0,count2);                                        /*column totals*/
            expctd=vector(0,((count1+1)*(count2+1)-1));                      /*expected frequen
cies*/


        /***************************************/
        /*Statistical Analysis of crosstab matrix*/
        /***************************************/

            estprob = MEANINGLESS;  /*meaningless, unless determined */
            *prob = MEANINGLESS;    /*meaningless, unless determined */


            if (count1 >= 1 && count2 >= 1) {           /*tests that matrix is min. of 2x2
*/

                test_flag = cochrantest(chi_matrix, rowtot, coltot, expctd, count1+1, coun
t2+1); /*cochran test*/


                if (test_flag == 0 || test_flag == 1)   {   /*do normal chi_square test wi
th estimated probability*/

                    chi_matx2 = subimatrix(chi_matrix,0,POSITIONS,0,POSITIONS,ORIGIN,ORIGI
N);

/*shifts table to ORIGIN,ORIGIN*/
                    chisquare(chi_matx2, count1+ORIGIN, count2+ORIGIN, chisq, df, prob, cr
amrv, ccc);

/*determines chi, est. prob, cram*/
                    free_subimatrix(chi_matx2, 1, POSITIONS+ORIGIN, 1, POSITIONS+ORIGIN);
/*deallocate memory*/

                    estprob = *prob;
                }

                if ((test_flag == 1)  &&  (estprob != MEANINGLESS)) {        /*do exact or
 estimated exact probability determination*/
                                                    /*NOTE this second condition is b/c in
 chisquare, gap check of 50%*/

                    matrix_by_rows = ivector(0,((count1+1)*(count2+1)-1));           /*
allocate memory*/
                    matrixconverter(chi_matrix, matrix_by_rows, count1, count2, 0);     /*
converts 2d matrix to 1d by rows matrix*/
                    estprob = estExact(matrix_by_rows, PRECISION, count1+1, count2+1);  /*
determine est. exact prob*/
                    free_ivector(matrix_by_rows,0,((count1+1)*(count2+1)-1));
      /*deallocate memory*/
                }

                else  { *prob = MEANINGLESS; estprob = MEANINGLESS; }
            }

            else  { *prob = MEANINGLESS; estprob = MEANINGLESS;  }


        /************************/
        /* LOG conversion       **/
        /************************/
```

```
            if (estprob > 0.0)  logprob = (-1)*log10(estprob);
            else logprob = 0.0;


        /*************************/
        /** OUTPUT to files     **/      /***format: column1, column2, Chi, prob, cram ***
*/
        /*************************/


            if ((pr < 0.0) || (i==j || estprob <0.0) || estprob == 1.0) {
                fprintf(output2, "%d\t%d\t%.3f\t%.5f\t%.31f\t%.5f\t%f\t%.1f\n", i,j,chi,pr
,estprob,cram,d,logprob);
            }       /*NOTE: Precision of log prob is 1 place, because calculation of preci
sion is to 10 hits */
                    /*SEE function estExact (tally < 10) for further explaination */
            else  {


                    /*************************/
                    /* tally distributions.  */
                    /*************************/


                index = (int) (cram*1000);
                cramfreq[index]++;

                index2 = (int) (logprob*100);
                lpfreq[index2]++;

                    /*********************/


                fprintf(output, "%d\t%d\t%.3f\t%.5f\t%.201f\t%.5f\t%f\t%.1f\n", i,j,chi,pr
,estprob,cram,d,logprob);

            }


        /************************************/
        /**** Free memory for this matrix ****/
        /************************************/

            free_ivector(coltot,0,(count1));
            free_ivector(rowtot,0,(count2));
            free_vector(expctd,0,((count1+1)*(count2+1)-1));


/*****************************/
/**** Debugging Display ******/
/*****************************/

    /*** start of display ****/

/*
    printf("\n\n\n\nCrosstabulation matrix for %d x %d\n",m,n);
    printf("\nLabels incorrect due to gap rectification\n");

    for (iqw = 0; iqw <= count2; iqw++)
            printf("\t%c",unique2[iqw]);

    printf("\n");

    for (jqw = 0; jqw <= count1; jqw++) {
            printf("%c",unique1[jqw]);

        for (kqw = 0; kqw <= count2; kqw++) {
            printf("\t%d", chi_matrix[jqw][kqw]);
        }

        printf("\n");
    }

    printf("\nchi:%f df:%f prob:%f cram:%f cc:%f\n", chi, d, pr, cram, cc);
*/
```

```
    /*** END Of DISPLAY  ****/


/***********************************/
/***** END OF DEBUGGING DISPLAY ****/
/***********************************/


        }
    }


    /***************************************/
    /*** END TIMER ***************************/
    timer = GetFloatTimer (timer);
    fprintf(output, "\n\tTotal Elapsed Time is: %.5f\n",timer);
    fprintf(output, "Given: %f Est.Exact Iterations\n",PRECISION);
    fprintf(output, "Given: %f Columns and %d Rows",NUM_COL, NUM_ROW);
    /***************************************/



    /***************************************/
    /******* OUTPUT of Distributions    *****/

    if (prec*100 < 1000)      { /*V dist is finer than P distribution */
        for (ww = 0; ww<=prec*100; ww++)
        {
            fprintf(output3, "%d\t%d\t%d\t%d\n",ww,cramfreq[ww],ww,lpfreq[ww]);
        }

        for (;ww < 1001; ww++)
        {
            fprintf(output3, "%d\t%d\t%d\t%d\n",ww,cramfreq[ww],NEGONE,NEGONE);
        }

        *dlength = ww -1;
    }


    else if (prec*100 > 1000)     { /*V dist is less fine than P distribution */
        for (ww = 0; ww< 1001; ww++)
        {
            fprintf(output3, "%d\t%d\t%d\t%d\n",ww,cramfreq[ww],ww,lpfreq[ww]);
        }

        for (;ww <= prec*100; ww++)
        {
            fprintf(output3, "%d\t%d\t%d\t%d\n",NEGONE,NEGONE,ww,lpfreq[ww]);
        }

        *dlength = ww -1;
    }

    else {

        for (ww = 0; ww< 1001; ww++)
        {
            fprintf(output3, "%d\t%d\t%d\t%d\n",ww,cramfreq[ww],ww,lpfreq[ww]);
        }

        *dlength = ww -1;
    }

    /***************************************/
    /***************************************/



    /****close output data pointer file****/
    fclose(output);
    fclose(output2);
```

```
    fclose(output3);
    /************************************/

    return chi_matrix;    /*return pointer for sole purpose of freeing memory allocation*/

}



/************************************************/
/********FUNCTION: find_unique_elements************/
/************************************************/

int find_unique_elements(char *input, char *aa, int *gap, int var)
{

    extern int  OFFSET;
    extern int  NUM_ROW;

    int i;
    int j;
    int k;
    int r;
    int ext;
    int counter = 0;        /* # of unique amino acids */


    for (k = 0; k < POSITIONS; k++)  {        /*clear buffer*/
        aa[k] = 0;
    }


    *gap = -1;              /* start off = -1; signifies no gap in position, in any sequen
ce*/

    aa[0] = *(input + var);     /*the first amino acid is always unique*/



    for (i = 1; i < NUM_ROW; i++) {         /*now compare REST of aa=NR-1 :therefore < */
        j = counter;
        ext = NO;

        while (ext == NO) {
            if (*(input + (i * OFFSET + var)) != aa[j]) {

                j = j - 1;
                if (j < 0)      /* No Match, therefore unique */ {
                    counter = counter + 1;
                    aa[counter] = *(input + (i * OFFSET + var));
                    ext = YES;
                }
                else /* No match yet, check previous elements */
                ext = NO;
            }
            else /*MATCHED, so exit*/
            ext = YES;
        }
    }


    for (r = 0; r<=counter; r++) {
        if (aa[r] == '-')           {
            *gap = r;
        }
    }

    return counter;
}


/************************************************/
/********FUNCTION: Crosstab **********************/
```

```
/************************************************/

/* Description: using two lists of unique amino acids, found in position var1, var2 */
/* compare all aa in a position to the list [0...p or q]                             */
/* when a match is found in position var, add 1 to matrix position p                 */
/* when a match is found in position var2, add 1 to matrix position q                */
/* Thus, chi_matrix[p][q] is a tally of frequency of all possible combinations       */
/* of amino acids found in var1 with var 2,                                          */


void    crosstab(char *input, int **chi_matrix, char *aa1, char *aa2, int var1, int var2,
int count1, int count2)
{
    extern int  OFFSET;
    extern int  NUM_ROW;

    void exitprogram();

    int i, j, k, n, m, p, q;
    int found_p = 0;
    int found_q = 0;

    for (j = 0; j <= POSITIONS; j++) {              /*initialize all values to zero */
        for (k = 0; k <= POSITIONS; k++)   {
            chi_matrix[j][k] = 0;
        }
    }

    for (i = 0; i < NUM_ROW; i++)    {

        n = 0;                          /*n are unique aa in 1*/
        /*input compares to n*/
        m = 0;                          /*m are unique aa in 2*/
        /*input compares to m*/
        found_p = NO;                   /*when a match is found, signal*/
        found_q = NO;                   /*when a match is found, signal*/


        while (found_p == NO)   {
            if ( n > count1 )   {
                printf("ERROR: overflow of unique characters set 1");
                printf("\n n = %d; i = %d",n,i);
                printf("\n %s = aa1 after overflow", aa1);
                printf("\n%c = input",(*input+(i*OFFSET+var1)));
                exitprogram();
                exit(4);
            }
            if (  *(input + (i * OFFSET + var1))  == aa1[n] )   {
                p = n;
                found_p = YES;
            }
            else {
                n += 1;
                found_p = NO;
            }
        }

        while (found_q == NO)   {
            if (m > count2 )     {
                printf("ERROR: overflow of unique characters set 2");
                exitprogram();
                exit(3);
            }
            if (  *(input + (i * OFFSET + var2))   == aa2[m] )   {
                q = m;
                found_q = YES;
            }
            else {
                m += 1;
                found_q = NO;
            }
        }

        chi_matrix[p][q] += 1;
```

```
    }
}



/*************************************************/
/**********FUNCTION: gap_rectifier ***************/
/*************************************************/


void gap_rectifer(int **chi_matrix,int gap1, int gap2,int *cnt1, int *cnt2)


/* removes gap (-) from chi_matrix   */
/* NOTE: any zero columns and rows   */
/* produced by gap-as-value removal, */
/* removed in chisq funct.           */

{

    int i,j,k,l;

    if (gap1 != -1)  {


        for (i = (gap1+1); i <= *cnt1 ; i++ )   {

            for (j = 0; j <= *cnt2; j++)            {
                chi_matrix[i-1][j] = chi_matrix[i][j];      /*removes gap values*/
            }
        }

        *cnt1 = *cnt1 - 1;

    }


    if (gap2 != -1) {

        for (k = (gap2+1); k <= *cnt2 ; k++ )   {

            for (l = 0; l <= *cnt1; l++)            {
                chi_matrix[l][k-1] = chi_matrix[l][k];      /*removes gap values*/
            }
        }

        *cnt2 = *cnt2 - 1;

    }


}



/*************************************************/
/**********FUNCTION: matrixconverter **************/
/*************************************************/

void matrixconverter(int **chi_matrix, int *matrix, int count1, int count2, int flag)
{

    int i,j;
    int index;

    if (flag == 0)  {

        for (i = 0; i <= count1; i++)   {
            for (j = 0; j <= count2; j++) {
                index = i*(count2+1) + j;
                matrix[index] = chi_matrix[i][j];
            }
```

```
        }


    }
    else if (flag == 1) {

        for (j = 0; j <= count2; j++)    {
            for (i = 0; i <= count1; i++) {
                index = j*(count1+1) + i;
                matrix[index] = chi_matrix[i][j];
            }
        }

    }

}




/*****************************************/
/**** cochrantest FUNCTION ***************/
/*****************************************/



int cochrantest(int **chi_matrix, int *rowtot, int *coltot, float *expctd, int numrows, in
t numcols)
{

float tally;
int numrows2,numcols2,i,j,r;
float sum=0.0;


    /***********************/
    /*clears memory = 0    */
    /***********************/


for (r=0;r < numrows*numcols; r++) {
    *(expctd + r) = 0;                          /*can probably combine with step below*/
}

    /***********************/
    /*calc row and col info*/
    /***********************/

numrows2=numrows;                               /*Number of rows*/
numcols2=numcols;                               /*and columns.*/

for (i=0;i< numrows;i++) {                      /*Get the row totals.*/
    rowtot[i]=0;                                /*clears memory*/
    for (j=0;j< numcols;j++) {
        rowtot[i] += chi_matrix[i][j];
        sum += chi_matrix[i][j];
    }
    if (rowtot[i] == 0) --numrows2;             /*Eliminate any zero rows by reducing the
num*/
}


for (j=0;j< numcols;j++) {                      /*Get the column totals.*/
    coltot[j]=0;                                /*clears memory*/
    for (i=0;i< numrows;i++) coltot[j] += chi_matrix[i][j];
    if (coltot[j] == 0) --numcols2;             /*Eliminate any zero columns.*/
}


    /*********************************************************/
    /*** Test of Matrix: must be 2x2 after zero removal *******/
    /*********************************************************/

if ((numcols2 < 2) || (numrows2 < 2))    {      /** Chisquare statistical analysis not pos
sible **/
```

```
    return MEANINGLESS;
}


    /*****************************************************/
    /* Calculate expected frequencies of original Rmatrix*/
    /*****************************************************/

tally = 0.0;

for (i=0;i< numrows;i++) {                    /*Do the chi-square sum.*/
    for (j=0;j< numcols;j++) {
        *(expctd +(i*numcols + j)) = (coltot[j]*rowtot[i])/sum;

        if (*(expctd +(i*numcols + j)) >= EXPECTED)
            tally += 1;

        else if (*(expctd +(i*numcols + j)) <= MINIMUM)
            return 1;

    }
}


if ( (tally/sum) < PERCENT ) return 1;

else return 0;


}



/*****************************************/
/**** exitprogram FUNCTION *****************/
/*****************************************/
void exitprogram() {

    float exitprog;
    printf("\n\nPROGRAM EXECUTION COMPLETED");
    printf("\n\n\nPlease enter '1.1' to exit program:");
    scanf("%f",&exitprog);


    printf("GOOD BYE");

}
/*****************************************/
/**** postionrelater FUNCTION ***************/
/*****************************************/


void positionrelater(char *input)
{

    int i,j;


    char    outfile[30] = "Position";       /*output file for data*/
    char    fullname[40];                       /*output file for data*/

    extern float NUM_COL;
    extern int OFFSET;
    extern int mainseq;



    FILE * output = NULL;                       /*output file initiation*/
    sprintf(fullname, "%s.txt", outfile);

    output = fopen(fullname, "w");

    if (output == NULL)      {
        printf("\nFailure to Create Position File.\n");
        exit(2);
```

```
    }

    mainseq = mainseq - 1;  /*because sequences are in data input starting at zero*/
    j = 1;                  /*1st character is 1*/

    for(i=0;i < NUM_COL; i++) {

        if ( (*(input + (mainseq * OFFSET + i)) ) != '-' )  {

            fprintf(output, "%d\t%d\t%c\n",i,j,(*(input + (mainseq * OFFSET + i))) );
            j = j+1;
        }

    }


    fclose(output);


}


/*******************************************/
/************  MESSAGE  ********************/
/*******************************************/

void    message(int number)
{

   int  numread;
   int  numwrite;
   char buf[10];
   FILE *inputfile2 = NULL;


   if (number == 1)  inputfile2 = fopen("scoremsg", "r");/* SCORING MESSAGE */
   if (number == 2)  inputfile2 = fopen("threshmsg", "r");/* Threshold uses and warnings M
ESSAGE */
   if (number == 3)  inputfile2 = fopen("predictmsg", "r");/* Predicition complete MESSAGE
 */
   if (number == 4)  inputfile2 = fopen("eliminatemsg", "r");/* eliminate intersecting int
eractions MESSAGE */
   if (number == 5)  inputfile2 = fopen("misalignmsg", "r");/* misalignment alg. intro MES
SAGE */
   if (number == 6)  inputfile2 = fopen("degenmsg", "r"); /* degenerate predictions MESSAG
E */

   /* opens input file */

    if (inputfile2 == NULL) printf("\nTEXT NOT FOUND: message %d\n",number);

    else {
        /*read FILE and writes to screen until end of file is reached*/

        while(  !feof(inputfile2)   )
        {
          numread =  fread(buf,sizeof(char),1,inputfile2);
          numwrite = fwrite(buf,sizeof(char),1,stdout);
        }

        fflush(inputfile2);
        fclose(inputfile2);
    }
}


/***************************/
/*!!!! END OF PROGRAM  !!!!!*/
/***************************/
```

```
/*******************************************************/
/********       Written By Phillip S. Pang      ********/
/*******************************************************/

/*******************************************************/
/******** MD/PhD Candidate, Columbia University ********/
/******** College of Physicians and Surgeons    ********/
/******** Dept. Of Biochemistry and Biophysics  ********/
/*******************************************************/

/*******************************************************/
/********    phillip.pang@stanfordalumni.org    ********/
/*******************************************************/


/*******************************************************/
/********       STATEMENT OF COPYRIGHT          ********/

/*          Copyright 2001 by The Trustees of          */
/*          Columbia University in the City of         */
/*          New York. ALL RIGHTS RESERVED;             */

/*******************************************************/


/*************************************************************/
/*************************************************************/
/*************************************************************/
/*                                                           */
/*  Certain algorithms found within this file may be derivatives*/
/*  of source code obtained from the book:                   */
/*  "Numerical Recipes in C: The Art of Scientific Computing"   */
/*  published by Cambridge University Press.                 */
/*                                                           */
/*************************************************************/
/*************************************************************/
/*************************************************************/


#include <math.h>

#include "nrutilp.h"
#include "definitions.h"



/*************************************************************/
/************   CHISQUARE function     ***********************/
/*************************************************************/

int chisquare(int **nn, int ni, int nj, float *chisq, float *df, double *prob, double *cra
mrv, float *ccc)

/*Given a two-dimensional contingency table in the form of an integer array nn[1..ni][1..n
j],
this routine returns the chi-square chisq, the number of degrees of freedom df, the signi_
cance
level prob (small values indicating a signi_cant association), and two measures of associa
tion,
Cramer's V (cramrv) and the contingency coe_cient C (ccc).  */

/*Since you pass the address, and fill the value, then of course, the address you passed,
will have a value in it!*/
/*this is how multiple values are returned! */

{
double gammq(double a, double x);
int nnj,nni,j,i,minij;
float sum=0.0,*sumi,*sumj;

double expctd, temp;

extern float CRV_CUTOFF;
extern int NUM_ROW;

sumi=vector(1,ni);
```

```
sumj=vector(1,nj);
nni=ni;                                 /*Number of rows*/
nnj=nj;                                 /*and columns.*/

for (i=1;i<=ni;i++) {                       /*Get the row totals.*/
    sumi[i]=0.0;
    for (j=1;j<=nj;j++) {
        sumi[i] += nn[i][j];
        sum += nn[i][j];
    }
    if (sumi[i] == 0.0) --nni;              /*Eliminate any zero rows by reducing the num*/
}


for (j=1;j<=nj;j++) {                       /*Get the column totals.*/
    sumj[j]=0.0;
    for (i=1;i<=ni;i++) sumj[j] += nn[i][j];
    if (sumj[j] == 0.0) --nnj;              /*Eliminate any zero columns.*/
}



/*************************************************************/
/*** 2nd Test of Matrix: must be 2x2 after zero removal ***/
/*************************************************************/

if ((nni < 2) || (nnj < 2)) {       /** Chisquare statistical analysis not possible **/
    *prob = MEANINGLESS;
    return 0;
}


/*************************************************************/
/*****************  End of 2nd Test **************************/
/*************************************************************/



*df= (float) (nni*nnj-nni-nnj+1);                   /*Corrected number of degrees of freed
om.*/
*chisq=0.0;

for (i=1;i<=ni;i++) {                    /*Do the chi-square sum.*/
    for (j=1;j<=nj;j++) {
        expctd=sumj[j]*sumi[i]/sum;
        temp=nn[i][j]-expctd;
        *chisq += (float) (temp*temp/(expctd+TINY));    /*Here TINY guarantees that any*/
/*elim. Div. By zero */
    }                               /* eliminated row or column will*/
}                                   /*not contribute to the sum.*/


*prob=gammq(0.5*(*df),0.5*(*chisq));        /*Chi-square probability function.*/
minij = nni < nnj ? nni-1 : nnj-1;
*cramrv=sqrt(*chisq/(sum*minij));
*ccc=(float) sqrt(*chisq/(*chisq+sum));



/********* TEST OF N versus SUM for cramer's V **********/

if ( sum < (NUM_ROW/2.0))   {   /*i.e. this matrix represents less then 50% of sequences *
/
        *cramrv = 0.0;
        *prob = MEANINGLESS;
}

/*************************************************************/


free_vector(sumj,1,nj);
free_vector(sumi,1,ni);

return 0;
}
```

```
/************************************************/
/********FUNCTION: gammq                 *********/
/************************************************/


double gammq(double a, double x)
{
    void gcf(double *gammcf, double a, double x, double *gln);
    void gser(double *gamser, double a, double x, double *gln);
    void nrerror(char error_text[]);
    double gamser,gammcf,gln;

    if (x < 0.0 || a <= 0.0)    /*nrerror("Invalid arguments in routine gammq");*/
            return MEANINGLESS;
    if (x < (a+1.0)) {
        gser(&gamser,a,x,&gln);
        return 1.0-gamser;
    } else {
        gcf(&gammcf,a,x,&gln);
        return gammcf;
    }
}


/************************************************/
/********FUNCTION: gcf                  *********/
/************************************************/


void gcf(double *gammcf, double a, double x, double *gln)
{
    double gammln(double xx);
    void nrerror(char error_text[]);
    int i;
    double an,b,c,d,del,h;

    *gln=gammln(a);
    b=x+1.0-a;
    c=1.0/FPMIN;
    d=1.0/b;
    h=d;
    for (i=1;i<=ITMAX;i++) {
        an = -i*(i-a);
        b += 2.0;
        d=an*d+b;
        if (fabs(d) < FPMIN) d=FPMIN;
        c=b+an/c;
        if (fabs(c) < FPMIN) c=FPMIN;
        d=1.0/d;
        del=d*c;
        h *= del;
        if (fabs(del-1.0) < EPS) break;
    }
    if (i > ITMAX) nrerror("a too large, ITMAX too small in gcf");
    *gammcf=exp(-x+a*log(x)-(*gln))*h;
}


/************************************************/
/********FUNCTION: gser                 *********/
/************************************************/

void gser(double *gamser, double a, double x, double *gln)
{
    double gammln(double xx);
    void nrerror(char error_text[]);
    int n;
    double sum,del,ap;

    *gln=gammln(a);
    if (x <= 0.0) {
```

```
        if (x < 0.0) nrerror("x less than 0 in routine gser");
        *gamser=0.0;
        return;
    } else {
        ap=a;
        del=sum=1.0/a;
        for (n=1;n<=ITMAX;n++) {
            ++ap;
            del *= x/ap;
            sum += del;
            if (fabs(del) < fabs(sum)*EPS) {
                *gamser=sum*exp(-x+a*log(x)-(*gln));
                return;
            }
        }
        nrerror("a too large, ITMAX too small in routine gser");
        return;
    }
}

/*************************************************/
/********FUNCTION: gammln                ***********/
/*************************************************/

double gammln(double xx)
{
    double x,y,tmp,ser;
    static double cof[6]={76.18009172947146,-86.50532032941677,
        24.01409824083091,-1.231739572450155,
        0.1208650973866179e-2,-0.5395239384953e-5};
    int j;

    y=x=xx;
    tmp=x+5.5;
    tmp -= (x+0.5)*log(tmp);
    ser=1.000000000190015;
    for (j=0;j<=5;j++) ser += cof[j]/++y;
    return -tmp+log(2.5066282746310005*ser/x);
}
```

```
/*******************************************************/
/********from Numerical Recipies in C  ************/
/*******************************************************/

/*********************************************************/
/********       Written By Phillip S. Pang       ********/
/*********************************************************/

/*********************************************************/
/******** MD/PhD Candidate, Columbia University ********/
/******** College of Physicians and Surgeons    ********/
/******** Dept. Of Biochemistry and Biophysics  ********/
/*********************************************************/

/*********************************************************/
/********    phillip.pang@stanfordalumni.org    ********/
/*********************************************************/


/*********************************************************/
/********       STATEMENT OF COPYRIGHT          ********/

/*          Copyright 2001 by The Trustees of          */
/*          Columbia University in the City of         */
/*          New York. ALL RIGHTS RESERVED;             */

/*********************************************************/


/*****************************************************************/
/*****************************************************************/
/*****************************************************************/
/*                                                               */
/*  Certain algorithms found within this file may be derivatives*/
/*  of source code obtained from the book:                       */
/*  "Numerical Recipes in C: The Art of Scientific Computing"    */
/*  published by Cambridge University Press.                     */
/*                                                               */
/*****************************************************************/
/*****************************************************************/
/*****************************************************************/


#include <stdio.h>
#include <stddef.h>
#include <stdlib.h>

#define NR_END 1
#define FREE_ARG char*


void nrerror(char error_text[])
/* Numerical Recipes standard error handler */
{
    fprintf(stderr,"Numerical Recipes run-time error...\n");
    fprintf(stderr,"%s\n",error_text);
    fprintf(stderr,"...now exiting to system...\n");
    exit(1);
}


double *dvector(long nl, long nh)
/* allocate a DOUBLE vector with subscript range v[nl..nh] */
{
    double *v;

    v=(double *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(double)));
    if (!v) {
        printf("FAILURE\n");
        exit(3);
    }
    return v-nl+NR_END;
}


void free_dvector(double *v, long nl, long nh)
```

```
/* free a float vector allocated with vector() */
{
    free((FREE_ARG) (v+nl-NR_END));
}


float *vector(long nl, long nh)
/* allocate a float vector with subscript range v[nl..nh] */
{
    float *v;

    v=(float *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(float)));
    if (!v) nrerror("allocation failure in vector()");
    return v-nl+NR_END;
}


int *ivector(long nl, long nh)
/* allocate an int vector with subscript range v[nl..nh] */
{
    int *v;

    v=(int *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(int)));
    if (!v) nrerror("allocation failure in vector()");
    return v-nl+NR_END;
}



unsigned char *cvector(long nl, long nh)
/* allocate an unsigned char vector with subscript range v[nl..nh] */
{
    unsigned char *v;

    v=(unsigned char *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(unsigned char)));
    if (!v) nrerror("allocation failure in cvector()");
    return v-nl+NR_END;
}


float **matrix(long nrl, long nrh, long ncl, long nch)
/* allocate a float matrix with subscript range m[nrl..nrh][ncl..nch] */
{
    long i, nrow=nrh-nrl+1,ncol=nch-ncl+1;
    float **m;

    /* allocate pointers to rows */
    m=(float **) malloc((size_t)((nrow+NR_END)*sizeof(float*)));
    if (!m) nrerror("allocation failure 1 in matrix()");
    m += NR_END;
    m -= nrl;

    /* allocate rows and set pointers to them */
    m[nrl]=(float *) malloc((size_t)((nrow*ncol+NR_END)*sizeof(float)));
    if (!m[nrl]) nrerror("allocation failure 2 in matrix()");
    m[nrl] += NR_END;
    m[nrl] -= ncl;

    for(i=nrl+1;i<=nrh;i++) m[i]=m[i-1]+ncol;

    /* return pointer to array of pointers to rows */
    return m;
}

int **imatrix(long nrl, long nrh, long ncl, long nch)
/* allocate a int matrix with subscript range m[nrl..nrh][ncl..nch] */
{
    long i, nrow=nrh-nrl+1,ncol=nch-ncl+1;
    int **m;

    /* allocate pointers to rows */
    m=(int **) malloc((size_t)((nrow+NR_END)*sizeof(int*)));
    if (!m) nrerror("allocation failure 1 in imatrix()");
    m += NR_END;
    m -= nrl;
```

```
    /* allocate rows and set pointers to them */
    m[nrl]=(int *) malloc((size_t)((nrow*ncol+NR_END)*sizeof(int)));
    if (!m[nrl]) nrerror("allocation failure 2 in imatrix()");
    m[nrl] += NR_END;
    m[nrl] -= ncl;

    for(i=nrl+1;i<=nrh;i++) m[i]=m[i-1]+ncol;

    /* return pointer to array of pointers to rows */
    return m;
}


int **subimatrix(int **a, long oldrl, long oldrh, long oldcl, long oldch,
    long newrl, long newcl)
/* point a submatrix [newrl..][newcl..] to a[oldrl..oldrh][oldcl..oldch] */
{
    long i,j,nrow=oldrh-oldrl+1,ncol=oldcl-newcl;
    int **m;

    /* allocate array of pointers to rows */
    m=(int **) malloc((size_t) ((nrow+NR_END)*sizeof(int*)));
    if (!m) nrerror("allocation failure in submatrix()");
    m += NR_END;
    m -= newrl;

    /* set pointers to rows */
    for(i=oldrl,j=newrl;i<=oldrh;i++,j++) m[j]=a[i]+ncol;

    /* return pointer to array of pointers to rows */
    return m;
}


float **convert_matrix(float *a, long nrl, long nrh, long ncl, long nch)
/* allocate a float matrix m[nrl..nrh][ncl..nch] that points to the matrix
declared in the standard C manner as a[nrow][ncol], where nrow=nrh-nrl+1
and ncol=nch-ncl+1. The routine should be called with the address
&a[0][0] as the first argument. */
{
    long i,j,nrow=nrh-nrl+1,ncol=nch-ncl+1;
    float **m;

    /* allocate pointers to rows */
    m=(float **) malloc((size_t) ((nrow+NR_END)*sizeof(float*)));
    if (!m) nrerror("allocation failure in convert_matrix()");
    m += NR_END;
    m -= nrl;

    /* set pointers to rows */
    m[nrl]=a-ncl;
    for(i=1,j=nrl+1;i<nrow;i++,j++) m[j]=m[j-1]+ncol;
    /* return pointer to array of pointers to rows */
    return m;
}


void free_vector(float *v, long nl, long nh)
/* free a float vector allocated with vector() */
{
    free((FREE_ARG) (v+nl-NR_END));
}


void free_ivector(int *v, long nl, long nh)
/* free a float vector allocated with vector() */
{
    free((FREE_ARG) (v+nl-NR_END));
}
```

```
void free_cvector(unsigned char *v, long nl, long nh)
/* free an unsigned char vector allocated with cvector() */
{
    free((FREE_ARG) (v+nl-NR_END));
}


void free_matrix(float **m, long nrl, long nrh, long ncl, long nch)
/* free a float matrix allocated by matrix() */
{
    free((FREE_ARG) (m[nrl]+ncl-NR_END));
    free((FREE_ARG) (m+nrl-NR_END));
}


void free_imatrix(int **m, long nrl, long nrh, long ncl, long nch)
/* free an int matrix allocated by imatrix() */
{
    free((FREE_ARG) (m[nrl]+ncl-NR_END));
    free((FREE_ARG) (m+nrl-NR_END));
}

void free_subimatrix(int **b, long nrl, long nrh, long ncl, long nch)
/* free a submatrix allocated by submatrix() */
{
    free((FREE_ARG) (b+nrl-NR_END));
}


void free_convert_matrix(float **b, long nrl, long nrh, long ncl, long nch)
/* free a matrix allocated by convert_matrix() */
{
    free((FREE_ARG) (b+nrl-NR_END));
}
```

```
/*****************************************************/
/********       Written By Phillip S. Pang      ********/
/*****************************************************/

/*****************************************************/
/******** MD/PhD Candidate, Columbia University ********/
/******** College of Physicians and Surgeons    ********/
/******** Dept. Of Biochemistry and Biophysics  ********/
/*****************************************************/

/*****************************************************/
/********    phillip.pang@stanfordalumni.org    ********/
/*****************************************************/


/*****************************************************/
/********       STATEMENT OF COPYRIGHT          ********/

/*          Copyright 2001 by The Trustees of          */
/*          Columbia University in the City of         */
/*          New York. ALL RIGHTS RESERVED;             */

/*****************************************************/



/*********************************************************************/
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/* THIS CODE WAS TRANSLATED, OPTIMIZED, and WRITTEN by RAUL Shakirov  */
/* from FORTRAN TO C, As a work-for-hire by request of Phillip S. Pang */
/*                                                                   */
/* Contact information for Raul: http://www.imach.uran.ru/rns/       */
/*                                                                   */
/*                                                                   */
/* It has been further modified by Phillip S. Pang                   */
/*                                                                   */
/*********************************************************************/
/*********************************************************************/
/*********************************************************************/



#include <math.h>       /* Header file for standard math functions */
#include <stdio.h>      /* Header file for standard io functions */
#include "fastexp2.h"   /* Header file for fastexp2 function */

/* ------------------------------------------------------------------
    Key constants (ref. explanations for function fastexp2 below).
------------------------------------------------------------------ */
#define EXPMINVAL (0)
#define EXPMAXVAL (8)
#define EXPBITS1  (9)
#define EXPBITS2  (11)

/* ------------------------------------------------------------------
    Derivative constants
    MULT1       pow (2, EXPBITS1)
    MULT2       pow (2, EXPBITS2)
    EXPTABLE1   num of elements in exptable1 (ref. fastexp2 below)
    EXPTABLE2   num of elements in exptable2 (ref. fastexp2 below)
    nmult12     multiplicator for fastexp2
------------------------------------------------------------------ */
#define MULT1      (2 << (EXPBITS1-1))
#define MULT2      (2 << (EXPBITS2-1))
#define EXPTABLE1 (MULT1 * EXPMAXVAL)
#define EXPTABLE2 (MULT2)
const float nmult12 = (- MULT1 * MULT2);

/* ------------------------------------------------------------------
    Include file with content of tables exptable1 and exptable2.
    This file is built by call of fastexp2inc (ref. below).
------------------------------------------------------------------ */
#include "fastexp2.inc"
```

```
/* ---------------------------------------------------------------------
    Name:       fastexp2ini

    Purpose:    Build tables for fastexp2 for given key constants.

    Usage:      fastexp2ini()

    Note:       Used for development purposes only!
--------------------------------------------------------------------- */

void fastexp2ini (void)
{
    int i;

    for (i = 0; i < EXPTABLE1; i++)
    {
        exptable1 [i] = (float)(exp (-((double)i) / MULT1));
    }

    for (i = 0; i < EXPTABLE2; i++)
    {
#if defined(_MSC_VER) && defined(_M_IX86)

        /*
            fastexp2 for VC++ and x86 processors uses coversion of
            double to nearest int (round mode) so we haven't add 0.5
            to exp argument here.
        */
        exptable2 [i] = (float)(exp (-((double)i) / (MULT1 * MULT2)));

#else /*_MSC_VER && _M_IX86*/

        /*
            fastexp2 for generic processors uses conversion of double
            to lower nearest int (floor mode) so we have add 0.5
            to exp argument for better approximation.
        */
        exptable2 [i] = (float)(exp (-((double)i+0.5) / (MULT1 * MULT2)));

#endif/*_MSC_VER && _M_IX86*/
    }
}

/* ---------------------------------------------------------------------
    Name:       fastexp2inc

    Purpose:    Print tables for fastexp2 for given key contatnts
                (file fastexp2.inc).

    Usage:      fastexp2inc()

    Note:       Used for development purposes only!
--------------------------------------------------------------------- */

void fastexp2inc (void)
{
    int i;

    printf ("/*\n"
        "    File with fastexp2 tables for case\n"
        "    EXPTABLE1 = %d, MULT1 = %d, EXPTABLE2 = %d, MULT2 = %d\n"
        "    The file was generated by function fastexp2inc.\n*/\n",
        EXPTABLE1, MULT1, EXPTABLE2, MULT2);

    printf ("\n#if EXPTABLE1 != %d || MULT1 != %d || "
                "EXPTABLE2 != %d || MULT2 != %d\n",
                 EXPTABLE1, MULT1, EXPTABLE2, MULT2);

    printf ("\nstatic float exptable1 [EXPTABLE1];");

    printf ("\nstatic float exptable2 [EXPTABLE2];");

    printf ("\n\n#else /*EXPTABLE && MULT*/\n");

    printf ("\nstatic float exptable1 [EXPTABLE1] =\n{");
```

```
    for (i = 0;;)
    {
        if (i % 4 == 0) printf ("\n   ");
        printf (" %.7ef",
                (float)(exp (-((double)i) / MULT1))
               );
        if (++i < EXPTABLE1) printf (","); else break;
    }

    printf ("\n};\n");

    printf ("\nstatic float exptable2 [EXPTABLE2] =\n{");

    printf ("\n#if defined(_MSC_VER) && defined(_M_IX86)\n");

    for (i = 0;;)
    {
        if (i % 4 == 0) printf ("\n   ");
        /*
            fastexp2 for VC++ and x86 processors uses coversion of
            double to nearest int (round mode) so we haven't add 0.5
            to exp argument here.
        */
        printf (" %.7ef",
                (float)(exp (-((double)i) / (MULT1 * MULT2)))
               );
        if (++i < EXPTABLE2) printf (","); else break;
    }

    printf ("\n\n#else /*_MSC_VER && _M_IX86*/\n");

    for (i = 0;;)
    {
        if (i % 4 == 0) printf ("\n   ");
        /*
            fastexp2 for generic processors uses conversion of double
            to lower nearest int (floor mode) so we have add 0.5
            to exp argument for better approximation.
        */
        printf (" %.7ef",
                (float)(exp (-((double)i+0.5) / (MULT1 * MULT2)))
               );
        if (++i < EXPTABLE2) printf (","); else break;
    }

    printf ("\n\n#endif/*_MSC_VER && _M_IX86*/\n};\n");

    printf ("\n#endif/*EXPTABLE && MULT*/\n");
}

/* ---------------------------------------------------------------------
    Name:       fastexp2

    Purpose:    Fast table-driven exponent algorithm.

    Usage:      fastexp2 (value)

    Result:     The exp of value.

    Method:     exp (i1 + i2) = exp (i1) * exp (i2)

                i1 is integer part of value and first EXPBITS1 bits
                of fractional part and i2 is successive EXPBITS2 bits
                of fractional part.

    Function gets exp (i1) from array exptable1 and exp (i2) from
    array exptable2 for values between -EXPMAXVAL and -EXPMINVAL.
    Max relative error is 1 / pow (2, EXPBITS1 + EXPBITS2 + 1).
    Function returns EXPBITS1 + EXPBITS2 + 1 significant bits.

    If value > -EXMINVAL or <= -EXMAXVAL, standard exp function is called.
--------------------------------------------------------------------- */

double fastexp2 (double value)
{
    register int i1, i2;                /* Components of value */
```

```
    /* Extract valid bits to integer variable */

#if defined(_MSC_VER) && defined(_M_IX86)

    /*
        fastexp2 for VC++ and x86 processors uses coversion of
        double to nearest int (round mode).
    */

    __asm fld   nmult12
    __asm fmul  value
    __asm fistp i1

#else /*_MSC_VER && _M_IX86*/

    /*
        fastexp2 for generic processors uses conversion of double
        to lower nearest int (floor mode).
        On x86 it's less efficient then conversion to nearest int.
    */

    i1 = (int) (value * nmult12);

#endif/*_MSC_VER && _M_IX86*/

    /* Divide valid bits to high and low parts */

    i2 = i1 & EXPTABLE2 - 1;            /* Get low  BITS2 bits */
    i1 >>= EXPBITS2;                    /* Get high BITS1 bits */

    if (i1 >= MULT1 * EXPMINVAL)
    {
        if (i1 < MULT1 * EXPMAXVAL)     /* EXPTABLE1 */
        {
            /* Use tables to get exp (i1) and exp (i2) */

            return (exptable1 [i1] * exptable2 [i2]);
        }
    }

    /* Use standard exp function. */

    return (exp (value));
}
```

```
/*******************************************************/
/********       Written By Phillip S. Pang      ********/
/*******************************************************/

/*******************************************************/
/******** MD/PhD Candidate, Columbia University ********/
/******** College of Physicians and Surgeons    ********/
/******** Dept. Of Biochemistry and Biophysics  ********/
/*******************************************************/

/*******************************************************/
/********    phillip.pang@stanfordalumni.org    ********/
/*******************************************************/


/*******************************************************/
/********       STATEMENT OF COPYRIGHT          ********/

/*          Copyright 2001 by The Trustees of          */
/*          Columbia University in the City of         */
/*          New York. ALL RIGHTS RESERVED;             */

/*******************************************************/



/*********************************************************************/
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/* THIS CODE WAS TRANSLATED, OPTIMIZED, and WRITTEN by RAUL Shakirov  */
/* from FORTRAN TO C, As a work-for-hire by request of Phillip S. Pang */
/*                                                                   */
/* Contact information for Raul: http://www.imach.uran.ru/rns/       */
/*                                                                   */
/*                                                                   */
/* It has been further modified by Phillip S. Pang                   */
/*                                                                   */
/*********************************************************************/
/*********************************************************************/
/*********************************************************************/


/*
    File with fastexp2 tables for case
    EXPTABLE1 = 4096, MULT1 = 512, EXPTABLE2 = 2048, MULT2 = 2048
    The file was generated by function fastexp2inc.
*/

#if EXPTABLE1 != 4096 || MULT1 != 512 || EXPTABLE2 != 2048 || MULT2 != 2048

static float exptable1 [EXPTABLE1];
static float exptable2 [EXPTABLE2];

#else /*EXPTABLE && MULT*/

static float exptable1 [EXPTABLE1] =
{
    1.0000000e+000f, 9.9804878e-001f, 9.9610137e-001f, 9.9415776e-001f,
    9.9221794e-001f, 9.9028190e-001f, 9.8834965e-001f, 9.8642116e-001f,
    9.8449644e-001f, 9.8257547e-001f, 9.8065825e-001f, 9.7874477e-001f,
    9.7683502e-001f, 9.7492901e-001f, 9.7302671e-001f, 9.7112812e-001f,
    9.6923323e-001f, 9.6734205e-001f, 9.6545455e-001f, 9.6357074e-001f,
    9.6169060e-001f, 9.5981413e-001f, 9.5794133e-001f, 9.5607217e-001f,
    9.5420667e-001f, 9.5234480e-001f, 9.5048657e-001f, 9.4863196e-001f,
    9.4678097e-001f, 9.4493359e-001f, 9.4308982e-001f, 9.4124965e-001f,
    9.3941306e-001f, 9.3758006e-001f, 9.3575064e-001f, 9.3392478e-001f,
    9.3210249e-001f, 9.3028376e-001f, 9.2846857e-001f, 9.2665692e-001f,
    9.2484881e-001f, 9.2304423e-001f, 9.2124317e-001f, 9.1944562e-001f,
    9.1765158e-001f, 9.1586104e-001f, 9.1407400e-001f, 9.1229044e-001f,
    9.1051036e-001f, 9.0873376e-001f, 9.0696062e-001f, 9.0519094e-001f,
    9.0342471e-001f, 9.0166193e-001f, 8.9990259e-001f, 8.9814669e-001f,
    8.9639421e-001f, 8.9464515e-001f, 8.9289950e-001f, 8.9115725e-001f,
    8.8941841e-001f, 8.8768296e-001f, 8.8595090e-001f, 8.8422221e-001f,
    8.8249690e-001f, 8.8077496e-001f, 8.7905637e-001f, 8.7734114e-001f,
    8.7562926e-001f, 8.7392071e-001f, 8.7221550e-001f, 8.7051362e-001f,
```

```
8.6881506e-001f, 8.6711981e-001f, 8.6542787e-001f, 8.6373923e-001f,
8.6205388e-001f, 8.6037183e-001f, 8.5869305e-001f, 8.5701756e-001f,
8.5534533e-001f, 8.5367636e-001f, 8.5201065e-001f, 8.5034819e-001f,
8.4868898e-001f, 8.4703300e-001f, 8.4538025e-001f, 8.4373073e-001f,
8.4208443e-001f, 8.4044134e-001f, 8.3880145e-001f, 8.3716477e-001f,
8.3553127e-001f, 8.3390097e-001f, 8.3227385e-001f, 8.3064990e-001f,
8.2902912e-001f, 8.2741150e-001f, 8.2579704e-001f, 8.2418573e-001f,
8.2257756e-001f, 8.2097253e-001f, 8.1937064e-001f, 8.1777186e-001f,
8.1617621e-001f, 8.1458367e-001f, 8.1299424e-001f, 8.1140791e-001f,
8.0982468e-001f, 8.0824453e-001f, 8.0666747e-001f, 8.0509349e-001f,
8.0352257e-001f, 8.0195473e-001f, 8.0038994e-001f, 7.9882820e-001f,
7.9726951e-001f, 7.9571386e-001f, 7.9416125e-001f, 7.9261167e-001f,
7.9106511e-001f, 7.8952157e-001f, 7.8798104e-001f, 7.8644352e-001f,
7.8490899e-001f, 7.8337746e-001f, 7.8184892e-001f, 7.8032336e-001f,
7.7880078e-001f, 7.7728117e-001f, 7.7576453e-001f, 7.7425084e-001f,
7.7274011e-001f, 7.7123232e-001f, 7.6972748e-001f, 7.6822557e-001f,
7.6672660e-001f, 7.6523054e-001f, 7.6373741e-001f, 7.6224719e-001f,
7.6075988e-001f, 7.5927547e-001f, 7.5779396e-001f, 7.5631534e-001f,
7.5483960e-001f, 7.5336674e-001f, 7.5189676e-001f, 7.5042965e-001f,
7.4896539e-001f, 7.4750400e-001f, 7.4604545e-001f, 7.4458976e-001f,
7.4313690e-001f, 7.4168688e-001f, 7.4023968e-001f, 7.3879531e-001f,
7.3735376e-001f, 7.3591502e-001f, 7.3447909e-001f, 7.3304596e-001f,
7.3161563e-001f, 7.3018809e-001f, 7.2876333e-001f, 7.2734135e-001f,
7.2592215e-001f, 7.2450572e-001f, 7.2309205e-001f, 7.2168114e-001f,
7.2027298e-001f, 7.1886757e-001f, 7.1746490e-001f, 7.1606497e-001f,
7.1466777e-001f, 7.1327330e-001f, 7.1188155e-001f, 7.1049251e-001f,
7.0910618e-001f, 7.0772256e-001f, 7.0634164e-001f, 7.0496341e-001f,
7.0358787e-001f, 7.0221502e-001f, 7.0084485e-001f, 6.9947734e-001f,
6.9811251e-001f, 6.9675034e-001f, 6.9539083e-001f, 6.9403397e-001f,
6.9267976e-001f, 6.9132819e-001f, 6.8997925e-001f, 6.8863295e-001f,
6.8728928e-001f, 6.8594823e-001f, 6.8460979e-001f, 6.8327397e-001f,
6.8194075e-001f, 6.8061014e-001f, 6.7928212e-001f, 6.7795669e-001f,
6.7663385e-001f, 6.7531359e-001f, 6.7399590e-001f, 6.7268079e-001f,
6.7136824e-001f, 6.7005825e-001f, 6.6875082e-001f, 6.6744594e-001f,
6.6614361e-001f, 6.6484382e-001f, 6.6354656e-001f, 6.6225184e-001f,
6.6095964e-001f, 6.5966996e-001f, 6.5838280e-001f, 6.5709815e-001f,
6.5581601e-001f, 6.5453637e-001f, 6.5325923e-001f, 6.5198458e-001f,
6.5071241e-001f, 6.4944273e-001f, 6.4817552e-001f, 6.4691079e-001f,
6.4564853e-001f, 6.4438872e-001f, 6.4313138e-001f, 6.4187649e-001f,
6.4062405e-001f, 6.3937405e-001f, 6.3812649e-001f, 6.3688137e-001f,
6.3563867e-001f, 6.3439840e-001f, 6.3316055e-001f, 6.3192512e-001f,
6.3069209e-001f, 6.2946148e-001f, 6.2823326e-001f, 6.2700744e-001f,
6.2578401e-001f, 6.2456297e-001f, 6.2334431e-001f, 6.2212803e-001f,
6.2091412e-001f, 6.1970258e-001f, 6.1849341e-001f, 6.1728659e-001f,
6.1608213e-001f, 6.1488002e-001f, 6.1368025e-001f, 6.1248283e-001f,
6.1128774e-001f, 6.1009498e-001f, 6.0890455e-001f, 6.0771645e-001f,
6.0653066e-001f, 6.0534719e-001f, 6.0416602e-001f, 6.0298716e-001f,
6.0181060e-001f, 6.0063634e-001f, 5.9946436e-001f, 5.9829468e-001f,
5.9712727e-001f, 5.9596215e-001f, 5.9479929e-001f, 5.9363871e-001f,
5.9248039e-001f, 5.9132433e-001f, 5.9017053e-001f, 5.8901898e-001f,
5.8786967e-001f, 5.8672261e-001f, 5.8557779e-001f, 5.8443520e-001f,
5.8329484e-001f, 5.8215670e-001f, 5.8102078e-001f, 5.7988709e-001f,
5.7875560e-001f, 5.7762632e-001f, 5.7649924e-001f, 5.7537437e-001f,
5.7425169e-001f, 5.7313120e-001f, 5.7201289e-001f, 5.7089677e-001f,
5.6978282e-001f, 5.6867105e-001f, 5.6756145e-001f, 5.6645402e-001f,
5.6534874e-001f, 5.6424562e-001f, 5.6314465e-001f, 5.6204584e-001f,
5.6094916e-001f, 5.5985463e-001f, 5.5876223e-001f, 5.5767196e-001f,
5.5658382e-001f, 5.5549780e-001f, 5.5441391e-001f, 5.5333212e-001f,
5.5225245e-001f, 5.5117488e-001f, 5.5009942e-001f, 5.4902606e-001f,
5.4795479e-001f, 5.4688561e-001f, 5.4581851e-001f, 5.4475350e-001f,
5.4369057e-001f, 5.4262971e-001f, 5.4157092e-001f, 5.4051420e-001f,
5.3945954e-001f, 5.3840693e-001f, 5.3735638e-001f, 5.3630788e-001f,
5.3526143e-001f, 5.3421702e-001f, 5.3317464e-001f, 5.3213430e-001f,
5.3109599e-001f, 5.3005971e-001f, 5.2902544e-001f, 5.2799320e-001f,
5.2696297e-001f, 5.2593475e-001f, 5.2490854e-001f, 5.2388432e-001f,
5.2286211e-001f, 5.2184189e-001f, 5.2082366e-001f, 5.1980742e-001f,
5.1879317e-001f, 5.1778089e-001f, 5.1677058e-001f, 5.1576225e-001f,
5.1475589e-001f, 5.1375148e-001f, 5.1274904e-001f, 5.1174856e-001f,
5.1075002e-001f, 5.0975344e-001f, 5.0875880e-001f, 5.0776610e-001f,
5.0677533e-001f, 5.0578651e-001f, 5.0479961e-001f, 5.0381463e-001f,
5.0283158e-001f, 5.0185044e-001f, 5.0087122e-001f, 4.9989391e-001f,
4.9891851e-001f, 4.9794501e-001f, 4.9697341e-001f, 4.9600371e-001f,
4.9503590e-001f, 4.9406997e-001f, 4.9310593e-001f, 4.9214378e-001f,
4.9118350e-001f, 4.9022509e-001f, 4.8926855e-001f, 4.8831388e-001f,
4.8736108e-001f, 4.8641013e-001f, 4.8546104e-001f, 4.8451380e-001f,
4.8356840e-001f, 4.8262485e-001f, 4.8168315e-001f, 4.8074328e-001f,
```

```
4.7980524e-001f, 4.7886904e-001f, 4.7793466e-001f, 4.7700211e-001f,
4.7607137e-001f, 4.7514245e-001f, 4.7421534e-001f, 4.7329005e-001f,
4.7236655e-001f, 4.7144486e-001f, 4.7052497e-001f, 4.6960687e-001f,
4.6869057e-001f, 4.6777605e-001f, 4.6686332e-001f, 4.6595236e-001f,
4.6504319e-001f, 4.6413579e-001f, 4.6323016e-001f, 4.6232629e-001f,
4.6142419e-001f, 4.6052385e-001f, 4.5962527e-001f, 4.5872844e-001f,
4.5783336e-001f, 4.5694003e-001f, 4.5604844e-001f, 4.5515859e-001f,
4.5427047e-001f, 4.5338409e-001f, 4.5249944e-001f, 4.5161652e-001f,
4.5073531e-001f, 4.4985583e-001f, 4.4897806e-001f, 4.4810201e-001f,
4.4722766e-001f, 4.4635502e-001f, 4.4548409e-001f, 4.4461485e-001f,
4.4374731e-001f, 4.4288146e-001f, 4.4201730e-001f, 4.4115483e-001f,
4.4029404e-001f, 4.3943493e-001f, 4.3857750e-001f, 4.3772174e-001f,
4.3686765e-001f, 4.3601522e-001f, 4.3516446e-001f, 4.3431536e-001f,
4.3346791e-001f, 4.3262212e-001f, 4.3177798e-001f, 4.3093549e-001f,
4.3009464e-001f, 4.2925543e-001f, 4.2841786e-001f, 4.2758192e-001f,
4.2674762e-001f, 4.2591494e-001f, 4.2508389e-001f, 4.2425445e-001f,
4.2342664e-001f, 4.2260044e-001f, 4.2177586e-001f, 4.2095288e-001f,
4.2013151e-001f, 4.1931174e-001f, 4.1849357e-001f, 4.1767700e-001f,
4.1686202e-001f, 4.1604863e-001f, 4.1523683e-001f, 4.1442661e-001f,
4.1361797e-001f, 4.1281091e-001f, 4.1200543e-001f, 4.1120152e-001f,
4.1039917e-001f, 4.0959839e-001f, 4.0879918e-001f, 4.0800152e-001f,
4.0720542e-001f, 4.0641087e-001f, 4.0561788e-001f, 4.0482643e-001f,
4.0403652e-001f, 4.0324816e-001f, 4.0246133e-001f, 4.0167604e-001f,
4.0089229e-001f, 4.0011006e-001f, 3.9932936e-001f, 3.9855018e-001f,
3.9777252e-001f, 3.9699638e-001f, 3.9622175e-001f, 3.9544863e-001f,
3.9467703e-001f, 3.9390693e-001f, 3.9313833e-001f, 3.9237123e-001f,
3.9160563e-001f, 3.9084152e-001f, 3.9007890e-001f, 3.8931777e-001f,
3.8855813e-001f, 3.8779997e-001f, 3.8704328e-001f, 3.8628808e-001f,
3.8553434e-001f, 3.8478208e-001f, 3.8403129e-001f, 3.8328196e-001f,
3.8253409e-001f, 3.8178768e-001f, 3.8104273e-001f, 3.8029924e-001f,
3.7955719e-001f, 3.7881659e-001f, 3.7807743e-001f, 3.7733972e-001f,
3.7660345e-001f, 3.7586861e-001f, 3.7513521e-001f, 3.7440324e-001f,
3.7367270e-001f, 3.7294358e-001f, 3.7221589e-001f, 3.7148961e-001f,
3.7076476e-001f, 3.7004131e-001f, 3.6931928e-001f, 3.6859866e-001f,
3.6787944e-001f, 3.6716163e-001f, 3.6644522e-001f, 3.6573020e-001f,
3.6501658e-001f, 3.6430435e-001f, 3.6359352e-001f, 3.6288407e-001f,
3.6217600e-001f, 3.6146931e-001f, 3.6076401e-001f, 3.6006008e-001f,
3.5935752e-001f, 3.5865634e-001f, 3.5795652e-001f, 3.5725807e-001f,
3.5656098e-001f, 3.5586525e-001f, 3.5517088e-001f, 3.5447787e-001f,
3.5378620e-001f, 3.5309589e-001f, 3.5240692e-001f, 3.5171930e-001f,
3.5103302e-001f, 3.5034807e-001f, 3.4966447e-001f, 3.4898220e-001f,
3.4830125e-001f, 3.4762164e-001f, 3.4694336e-001f, 3.4626639e-001f,
3.4559075e-001f, 3.4491643e-001f, 3.4424342e-001f, 3.4357173e-001f,
3.4290134e-001f, 3.4223227e-001f, 3.4156450e-001f, 3.4089803e-001f,
3.4023286e-001f, 3.3956900e-001f, 3.3890642e-001f, 3.3824514e-001f,
3.3758515e-001f, 3.3692645e-001f, 3.3626903e-001f, 3.3561290e-001f,
3.3495804e-001f, 3.3430447e-001f, 3.3365217e-001f, 3.3300114e-001f,
3.3235138e-001f, 3.3170289e-001f, 3.3105566e-001f, 3.3040970e-001f,
3.2976500e-001f, 3.2912156e-001f, 3.2847937e-001f, 3.2783843e-001f,
3.2719875e-001f, 3.2656031e-001f, 3.2592312e-001f, 3.2528717e-001f,
3.2465247e-001f, 3.2401900e-001f, 3.2338677e-001f, 3.2275577e-001f,
3.2212600e-001f, 3.2149746e-001f, 3.2087015e-001f, 3.2024406e-001f,
3.1961920e-001f, 3.1899555e-001f, 3.1837312e-001f, 3.1775190e-001f,
3.1713190e-001f, 3.1651311e-001f, 3.1589552e-001f, 3.1527914e-001f,
3.1466396e-001f, 3.1404998e-001f, 3.1343720e-001f, 3.1282562e-001f,
3.1221523e-001f, 3.1160603e-001f, 3.1099801e-001f, 3.1039119e-001f,
3.0978555e-001f, 3.0918109e-001f, 3.0857781e-001f, 3.0797571e-001f,
3.0737478e-001f, 3.0677502e-001f, 3.0617644e-001f, 3.0557902e-001f,
3.0498277e-001f, 3.0438768e-001f, 3.0379375e-001f, 3.0320099e-001f,
3.0260937e-001f, 3.0201892e-001f, 3.0142961e-001f, 3.0084146e-001f,
3.0025445e-001f, 2.9966859e-001f, 2.9908387e-001f, 2.9850029e-001f,
2.9791785e-001f, 2.9733655e-001f, 2.9675638e-001f, 2.9617734e-001f,
2.9559944e-001f, 2.9502266e-001f, 2.9444700e-001f, 2.9387247e-001f,
2.9329906e-001f, 2.9272677e-001f, 2.9215560e-001f, 2.9158554e-001f,
2.9101659e-001f, 2.9044875e-001f, 2.8988202e-001f, 2.8931640e-001f,
2.8875188e-001f, 2.8818846e-001f, 2.8762614e-001f, 2.8706492e-001f,
2.8650480e-001f, 2.8594576e-001f, 2.8538782e-001f, 2.8483097e-001f,
2.8427520e-001f, 2.8372052e-001f, 2.8316691e-001f, 2.8261439e-001f,
2.8206295e-001f, 2.8151259e-001f, 2.8096329e-001f, 2.8041507e-001f,
2.7986792e-001f, 2.7932184e-001f, 2.7877682e-001f, 2.7823286e-001f,
2.7768997e-001f, 2.7714814e-001f, 2.7660736e-001f, 2.7606764e-001f,
2.7552897e-001f, 2.7499135e-001f, 2.7445478e-001f, 2.7391926e-001f,
2.7338479e-001f, 2.7285135e-001f, 2.7231896e-001f, 2.7178761e-001f,
2.7125729e-001f, 2.7072801e-001f, 2.7019976e-001f, 2.6967254e-001f,
2.6914635e-001f, 2.6862119e-001f, 2.6809705e-001f, 2.6757393e-001f,
2.6705184e-001f, 2.6653076e-001f, 2.6601070e-001f, 2.6549165e-001f,
```

```
2.6497362e-001f, 2.6445660e-001f, 2.6394059e-001f, 2.6342558e-001f,
2.6291158e-001f, 2.6239858e-001f, 2.6188659e-001f, 2.6137559e-001f,
2.6086559e-001f, 2.6035658e-001f, 2.5984857e-001f, 2.5934155e-001f,
2.5883551e-001f, 2.5833047e-001f, 2.5782641e-001f, 2.5732333e-001f,
2.5682124e-001f, 2.5632013e-001f, 2.5581999e-001f, 2.5532083e-001f,
2.5482264e-001f, 2.5432543e-001f, 2.5382918e-001f, 2.5333391e-001f,
2.5283960e-001f, 2.5234625e-001f, 2.5185387e-001f, 2.5136245e-001f,
2.5087198e-001f, 2.5038248e-001f, 2.4989393e-001f, 2.4940633e-001f,
2.4891968e-001f, 2.4843398e-001f, 2.4794924e-001f, 2.4746543e-001f,
2.4698257e-001f, 2.4650066e-001f, 2.4601968e-001f, 2.4553964e-001f,
2.4506054e-001f, 2.4458237e-001f, 2.4410514e-001f, 2.4362884e-001f,
2.4315346e-001f, 2.4267902e-001f, 2.4220550e-001f, 2.4173290e-001f,
2.4126123e-001f, 2.4079047e-001f, 2.4032064e-001f, 2.3985172e-001f,
2.3938372e-001f, 2.3891663e-001f, 2.3845045e-001f, 2.3798518e-001f,
2.3752082e-001f, 2.3705736e-001f, 2.3659481e-001f, 2.3613316e-001f,
2.3567242e-001f, 2.3521257e-001f, 2.3475362e-001f, 2.3429556e-001f,
2.3383840e-001f, 2.3338213e-001f, 2.3292675e-001f, 2.3247226e-001f,
2.3201866e-001f, 2.3156594e-001f, 2.3111410e-001f, 2.3066315e-001f,
2.3021307e-001f, 2.2976388e-001f, 2.2931556e-001f, 2.2886811e-001f,
2.2842154e-001f, 2.2797584e-001f, 2.2753101e-001f, 2.2708705e-001f,
2.2664395e-001f, 2.2620172e-001f, 2.2576035e-001f, 2.2531984e-001f,
2.2488019e-001f, 2.2444140e-001f, 2.2400347e-001f, 2.2356639e-001f,
2.2313016e-001f, 2.2269478e-001f, 2.2226026e-001f, 2.2182658e-001f,
2.2139375e-001f, 2.2096176e-001f, 2.2053062e-001f, 2.2010031e-001f,
2.1967085e-001f, 2.1924222e-001f, 2.1881443e-001f, 2.1838748e-001f,
2.1796136e-001f, 2.1753607e-001f, 2.1711160e-001f, 2.1668797e-001f,
2.1626517e-001f, 2.1584319e-001f, 2.1542203e-001f, 2.1500169e-001f,
2.1458218e-001f, 2.1416348e-001f, 2.1374560e-001f, 2.1332854e-001f,
2.1291229e-001f, 2.1249685e-001f, 2.1208222e-001f, 2.1166840e-001f,
2.1125539e-001f, 2.1084318e-001f, 2.1043178e-001f, 2.1002118e-001f,
2.0961139e-001f, 2.0920239e-001f, 2.0879419e-001f, 2.0838679e-001f,
2.0798018e-001f, 2.0757436e-001f, 2.0716934e-001f, 2.0676511e-001f,
2.0636166e-001f, 2.0595901e-001f, 2.0555714e-001f, 2.0515605e-001f,
2.0475574e-001f, 2.0435622e-001f, 2.0395748e-001f, 2.0355951e-001f,
2.0316232e-001f, 2.0276591e-001f, 2.0237027e-001f, 2.0197540e-001f,
2.0158130e-001f, 2.0118797e-001f, 2.0079541e-001f, 2.0040361e-001f,
2.0001258e-001f, 1.9962231e-001f, 1.9923281e-001f, 1.9884406e-001f,
1.9845607e-001f, 1.9806884e-001f, 1.9768237e-001f, 1.9729664e-001f,
1.9691168e-001f, 1.9652746e-001f, 1.9614399e-001f, 1.9576127e-001f,
1.9537930e-001f, 1.9499807e-001f, 1.9461758e-001f, 1.9423784e-001f,
1.9385884e-001f, 1.9348058e-001f, 1.9310306e-001f, 1.9272627e-001f,
1.9235022e-001f, 1.9197490e-001f, 1.9160032e-001f, 1.9122646e-001f,
1.9085334e-001f, 1.9048094e-001f, 1.9010927e-001f, 1.8973833e-001f,
1.8936811e-001f, 1.8899861e-001f, 1.8862983e-001f, 1.8826177e-001f,
1.8789443e-001f, 1.8752781e-001f, 1.8716190e-001f, 1.8679671e-001f,
1.8643223e-001f, 1.8606846e-001f, 1.8570540e-001f, 1.8534304e-001f,
1.8498140e-001f, 1.8462046e-001f, 1.8426023e-001f, 1.8390069e-001f,
1.8354186e-001f, 1.8318373e-001f, 1.8282630e-001f, 1.8246957e-001f,
1.8211353e-001f, 1.8175819e-001f, 1.8140354e-001f, 1.8104958e-001f,
1.8069631e-001f, 1.8034373e-001f, 1.7999184e-001f, 1.7964064e-001f,
1.7929012e-001f, 1.7894029e-001f, 1.7859113e-001f, 1.7824266e-001f,
1.7789487e-001f, 1.7754776e-001f, 1.7720133e-001f, 1.7685557e-001f,
1.7651048e-001f, 1.7616607e-001f, 1.7582234e-001f, 1.7547927e-001f,
1.7513687e-001f, 1.7479514e-001f, 1.7445408e-001f, 1.7411368e-001f,
1.7377394e-001f, 1.7343487e-001f, 1.7309646e-001f, 1.7275871e-001f,
1.7242162e-001f, 1.7208519e-001f, 1.7174942e-001f, 1.7141429e-001f,
1.7107983e-001f, 1.7074601e-001f, 1.7041285e-001f, 1.7008034e-001f,
1.6974847e-001f, 1.6941726e-001f, 1.6908669e-001f, 1.6875676e-001f,
1.6842748e-001f, 1.6809884e-001f, 1.6777084e-001f, 1.6744349e-001f,
1.6711677e-001f, 1.6679069e-001f, 1.6646524e-001f, 1.6614043e-001f,
1.6581626e-001f, 1.6549271e-001f, 1.6516980e-001f, 1.6484752e-001f,
1.6452586e-001f, 1.6420484e-001f, 1.6388444e-001f, 1.6356466e-001f,
1.6324551e-001f, 1.6292698e-001f, 1.6260908e-001f, 1.6229179e-001f,
1.6197513e-001f, 1.6165908e-001f, 1.6134364e-001f, 1.6102883e-001f,
1.6071463e-001f, 1.6040104e-001f, 1.6008806e-001f, 1.5977569e-001f,
1.5946393e-001f, 1.5915279e-001f, 1.5884224e-001f, 1.5853231e-001f,
1.5822298e-001f, 1.5791425e-001f, 1.5760612e-001f, 1.5729860e-001f,
1.5699168e-001f, 1.5668535e-001f, 1.5637962e-001f, 1.5607449e-001f,
1.5576996e-001f, 1.5546601e-001f, 1.5516267e-001f, 1.5485991e-001f,
1.5455774e-001f, 1.5425617e-001f, 1.5395518e-001f, 1.5365478e-001f,
1.5335497e-001f, 1.5305574e-001f, 1.5275709e-001f, 1.5245903e-001f,
1.5216155e-001f, 1.5186465e-001f, 1.5156833e-001f, 1.5127258e-001f,
1.5097742e-001f, 1.5068283e-001f, 1.5038881e-001f, 1.5009537e-001f,
1.4980250e-001f, 1.4951021e-001f, 1.4921848e-001f, 1.4892732e-001f,
1.4863673e-001f, 1.4834671e-001f, 1.4805725e-001f, 1.4776836e-001f,
1.4748003e-001f, 1.4719226e-001f, 1.4690506e-001f, 1.4661842e-001f,
```

```
1.4633233e-001f, 1.4604681e-001f, 1.4576184e-001f, 1.4547742e-001f,
1.4519356e-001f, 1.4491026e-001f, 1.4462751e-001f, 1.4434531e-001f,
1.4406366e-001f, 1.4378256e-001f, 1.4350201e-001f, 1.4322200e-001f,
1.4294255e-001f, 1.4266363e-001f, 1.4238527e-001f, 1.4210744e-001f,
1.4183016e-001f, 1.4155342e-001f, 1.4127722e-001f, 1.4100155e-001f,
1.4072643e-001f, 1.4045184e-001f, 1.4017779e-001f, 1.3990427e-001f,
1.3963129e-001f, 1.3935884e-001f, 1.3908692e-001f, 1.3881553e-001f,
1.3854467e-001f, 1.3827434e-001f, 1.3800453e-001f, 1.3773526e-001f,
1.3746650e-001f, 1.3719828e-001f, 1.3693057e-001f, 1.3666339e-001f,
1.3639673e-001f, 1.3613059e-001f, 1.3586497e-001f, 1.3559987e-001f,
1.3533528e-001f, 1.3507121e-001f, 1.3480766e-001f, 1.3454462e-001f,
1.3428210e-001f, 1.3402008e-001f, 1.3375858e-001f, 1.3349759e-001f,
1.3323710e-001f, 1.3297713e-001f, 1.3271766e-001f, 1.3245870e-001f,
1.3220024e-001f, 1.3194229e-001f, 1.3168484e-001f, 1.3142790e-001f,
1.3117145e-001f, 1.3091551e-001f, 1.3066007e-001f, 1.3040512e-001f,
1.3015067e-001f, 1.2989672e-001f, 1.2964326e-001f, 1.2939030e-001f,
1.2913783e-001f, 1.2888585e-001f, 1.2863437e-001f, 1.2838337e-001f,
1.2813287e-001f, 1.2788286e-001f, 1.2763333e-001f, 1.2738429e-001f,
1.2713573e-001f, 1.2688766e-001f, 1.2664008e-001f, 1.2639298e-001f,
1.2614635e-001f, 1.2590022e-001f, 1.2565456e-001f, 1.2540938e-001f,
1.2516468e-001f, 1.2492045e-001f, 1.2467671e-001f, 1.2443343e-001f,
1.2419064e-001f, 1.2394831e-001f, 1.2370646e-001f, 1.2346509e-001f,
1.2322418e-001f, 1.2298374e-001f, 1.2274377e-001f, 1.2250427e-001f,
1.2226524e-001f, 1.2202667e-001f, 1.2178857e-001f, 1.2155094e-001f,
1.2131376e-001f, 1.2107705e-001f, 1.2084081e-001f, 1.2060502e-001f,
1.2036969e-001f, 1.2013483e-001f, 1.1990042e-001f, 1.1966646e-001f,
1.1943297e-001f, 1.1919993e-001f, 1.1896734e-001f, 1.1873521e-001f,
1.1850353e-001f, 1.1827231e-001f, 1.1804153e-001f, 1.1781121e-001f,
1.1758133e-001f, 1.1735190e-001f, 1.1712293e-001f, 1.1689439e-001f,
1.1666631e-001f, 1.1643867e-001f, 1.1621147e-001f, 1.1598471e-001f,
1.1575840e-001f, 1.1553253e-001f, 1.1530710e-001f, 1.1508211e-001f,
1.1485756e-001f, 1.1463345e-001f, 1.1440978e-001f, 1.1418654e-001f,
1.1396373e-001f, 1.1374137e-001f, 1.1351943e-001f, 1.1329793e-001f,
1.1307686e-001f, 1.1285622e-001f, 1.1263602e-001f, 1.1241624e-001f,
1.1219689e-001f, 1.1197797e-001f, 1.1175948e-001f, 1.1154141e-001f,
1.1132377e-001f, 1.1110655e-001f, 1.1088976e-001f, 1.1067339e-001f,
1.1045744e-001f, 1.1024191e-001f, 1.1002681e-001f, 1.0981212e-001f,
1.0959785e-001f, 1.0938400e-001f, 1.0917057e-001f, 1.0895756e-001f,
1.0874496e-001f, 1.0853277e-001f, 1.0832100e-001f, 1.0810964e-001f,
1.0789870e-001f, 1.0768816e-001f, 1.0747804e-001f, 1.0726832e-001f,
1.0705902e-001f, 1.0685013e-001f, 1.0664164e-001f, 1.0643356e-001f,
1.0622588e-001f, 1.0601861e-001f, 1.0581175e-001f, 1.0560528e-001f,
1.0539922e-001f, 1.0519357e-001f, 1.0498831e-001f, 1.0478346e-001f,
1.0457900e-001f, 1.0437494e-001f, 1.0417129e-001f, 1.0396803e-001f,
1.0376516e-001f, 1.0356269e-001f, 1.0336062e-001f, 1.0315894e-001f,
1.0295765e-001f, 1.0275676e-001f, 1.0255626e-001f, 1.0235615e-001f,
1.0215643e-001f, 1.0195710e-001f, 1.0175816e-001f, 1.0155961e-001f,
1.0136144e-001f, 1.0116367e-001f, 1.0096627e-001f, 1.0076927e-001f,
1.0057264e-001f, 1.0037640e-001f, 1.0018055e-001f, 9.9985073e-002f,
9.9789980e-002f, 9.9595268e-002f, 9.9400936e-002f, 9.9206983e-002f,
9.9013408e-002f, 9.8820212e-002f, 9.8627392e-002f, 9.8434948e-002f,
9.8242880e-002f, 9.8051187e-002f, 9.7859867e-002f, 9.7668921e-002f,
9.7478348e-002f, 9.7288146e-002f, 9.7098316e-002f, 9.6908856e-002f,
9.6719765e-002f, 9.6531044e-002f, 9.6342691e-002f, 9.6154705e-002f,
9.5967086e-002f, 9.5779833e-002f, 9.5592946e-002f, 9.5406423e-002f,
9.5220264e-002f, 9.5034469e-002f, 9.4849036e-002f, 9.4663964e-002f,
9.4479254e-002f, 9.4294905e-002f, 9.4110915e-002f, 9.3927284e-002f,
9.3744011e-002f, 9.3561096e-002f, 9.3378538e-002f, 9.3196336e-002f,
9.3014489e-002f, 9.2832998e-002f, 9.2651860e-002f, 9.2471076e-002f,
9.2290645e-002f, 9.2110565e-002f, 9.1930838e-002f, 9.1751460e-002f,
9.1572433e-002f, 9.1393755e-002f, 9.1215426e-002f, 9.1037445e-002f,
9.0859811e-002f, 9.0682524e-002f, 9.0505582e-002f, 9.0328986e-002f,
9.0152734e-002f, 8.9976827e-002f, 8.9801262e-002f, 8.9626040e-002f,
8.9451160e-002f, 8.9276621e-002f, 8.9102423e-002f, 8.8928565e-002f,
8.8755046e-002f, 8.8581865e-002f, 8.8409023e-002f, 8.8236517e-002f,
8.8064348e-002f, 8.7892516e-002f, 8.7721018e-002f, 8.7549855e-002f,
8.7379026e-002f, 8.7208531e-002f, 8.7038368e-002f, 8.6868537e-002f,
8.6699037e-002f, 8.6529868e-002f, 8.6361030e-002f, 8.6192520e-002f,
8.6024340e-002f, 8.5856488e-002f, 8.5688963e-002f, 8.5521765e-002f,
8.5354893e-002f, 8.5188347e-002f, 8.5022126e-002f, 8.4856229e-002f,
8.4690656e-002f, 8.4525406e-002f, 8.4360479e-002f, 8.4195873e-002f,
8.4031588e-002f, 8.3867624e-002f, 8.3703980e-002f, 8.3540655e-002f,
8.3377649e-002f, 8.3214961e-002f, 8.3052591e-002f, 8.2890537e-002f,
8.2728799e-002f, 8.2567377e-002f, 8.2406270e-002f, 8.2245478e-002f,
8.2084999e-002f, 8.1924833e-002f, 8.1764980e-002f, 8.1605438e-002f,
8.1446208e-002f, 8.1287289e-002f, 8.1128679e-002f, 8.0970380e-002f,
```

8.0812389e-002f, 8.0654706e-002f, 8.0497331e-002f, 8.0340263e-002f,
8.0183502e-002f, 8.0027046e-002f, 7.9870896e-002f, 7.9715050e-002f,
7.9559509e-002f, 7.9404271e-002f, 7.9249336e-002f, 7.9094703e-002f,
7.8940372e-002f, 7.8786342e-002f, 7.8632612e-002f, 7.8479183e-002f,
7.8326053e-002f, 7.8173222e-002f, 7.8020689e-002f, 7.7868453e-002f,
7.7716515e-002f, 7.7564873e-002f, 7.7413527e-002f, 7.7262476e-002f,
7.7111720e-002f, 7.6961258e-002f, 7.6811090e-002f, 7.6661215e-002f,
7.6511632e-002f, 7.6362341e-002f, 7.6213341e-002f, 7.6064632e-002f,
7.5916214e-002f, 7.5768084e-002f, 7.5620244e-002f, 7.5472693e-002f,
7.5325429e-002f, 7.5178452e-002f, 7.5031763e-002f, 7.4885359e-002f,
7.4739242e-002f, 7.4593409e-002f, 7.4447861e-002f, 7.4302597e-002f,
7.4157616e-002f, 7.4012919e-002f, 7.3868503e-002f, 7.3724370e-002f,
7.3580517e-002f, 7.3436946e-002f, 7.3293654e-002f, 7.3150642e-002f,
7.3007909e-002f, 7.2865455e-002f, 7.2723278e-002f, 7.2581379e-002f,
7.2439757e-002f, 7.2298411e-002f, 7.2157341e-002f, 7.2016546e-002f,
7.1876026e-002f, 7.1735781e-002f, 7.1595808e-002f, 7.1456109e-002f,
7.1316683e-002f, 7.1177528e-002f, 7.1038645e-002f, 7.0900033e-002f,
7.0761692e-002f, 7.0623620e-002f, 7.0485818e-002f, 7.0348285e-002f,
7.0211020e-002f, 7.0074023e-002f, 6.9937293e-002f, 6.9800830e-002f,
6.9664634e-002f, 6.9528703e-002f, 6.9393037e-002f, 6.9257636e-002f,
6.9122499e-002f, 6.8987626e-002f, 6.8853016e-002f, 6.8718669e-002f,
6.8584583e-002f, 6.8450760e-002f, 6.8317198e-002f, 6.8183896e-002f,
6.8050854e-002f, 6.7918072e-002f, 6.7785549e-002f, 6.7653284e-002f,
6.7521278e-002f, 6.7389529e-002f, 6.7258038e-002f, 6.7126802e-002f,
6.6995823e-002f, 6.6865100e-002f, 6.6734631e-002f, 6.6604417e-002f,
6.6474458e-002f, 6.6344751e-002f, 6.6215298e-002f, 6.6086098e-002f,
6.5957149e-002f, 6.5828453e-002f, 6.5700007e-002f, 6.5571812e-002f,
6.5443867e-002f, 6.5316171e-002f, 6.5188725e-002f, 6.5061528e-002f,
6.4934579e-002f, 6.4807877e-002f, 6.4681423e-002f, 6.4555215e-002f,
6.4429254e-002f, 6.4303538e-002f, 6.4178068e-002f, 6.4052842e-002f,
6.3927861e-002f, 6.3803124e-002f, 6.3678630e-002f, 6.3554379e-002f,
6.3430371e-002f, 6.3306604e-002f, 6.3183079e-002f, 6.3059795e-002f,
6.2936752e-002f, 6.2813948e-002f, 6.2691384e-002f, 6.2569060e-002f,
6.2446974e-002f, 6.2325126e-002f, 6.2203516e-002f, 6.2082144e-002f,
6.1961008e-002f, 6.1840108e-002f, 6.1719445e-002f, 6.1599016e-002f,
6.1478823e-002f, 6.1358865e-002f, 6.1239140e-002f, 6.1119649e-002f,
6.1000391e-002f, 6.0881366e-002f, 6.0762573e-002f, 6.0644012e-002f,
6.0525682e-002f, 6.0407584e-002f, 6.0289715e-002f, 6.0172077e-002f,
6.0054668e-002f, 5.9937488e-002f, 5.9820537e-002f, 5.9703814e-002f,
5.9587319e-002f, 5.9471051e-002f, 5.9355010e-002f, 5.9239195e-002f,
5.9123607e-002f, 5.9008243e-002f, 5.8893105e-002f, 5.8778192e-002f,
5.8663503e-002f, 5.8549038e-002f, 5.8434796e-002f, 5.8320777e-002f,
5.8206980e-002f, 5.8093405e-002f, 5.7980053e-002f, 5.7866921e-002f,
5.7754010e-002f, 5.7641319e-002f, 5.7528848e-002f, 5.7416597e-002f,
5.7304564e-002f, 5.7192751e-002f, 5.7081155e-002f, 5.6969777e-002f,
5.6858617e-002f, 5.6747673e-002f, 5.6636946e-002f, 5.6526435e-002f,
5.6416140e-002f, 5.6306059e-002f, 5.6196194e-002f, 5.6086543e-002f,
5.5977106e-002f, 5.5867882e-002f, 5.5758872e-002f, 5.5650074e-002f,
5.5541488e-002f, 5.5433115e-002f, 5.5324953e-002f, 5.5217001e-002f,
5.5109261e-002f, 5.5001731e-002f, 5.4894410e-002f, 5.4787299e-002f,
5.4680397e-002f, 5.4573704e-002f, 5.4467219e-002f, 5.4360941e-002f,
5.4254871e-002f, 5.4149008e-002f, 5.4043351e-002f, 5.3937901e-002f,
5.3832656e-002f, 5.3727617e-002f, 5.3622783e-002f, 5.3518153e-002f,
5.3413727e-002f, 5.3309505e-002f, 5.3205487e-002f, 5.3101671e-002f,
5.2998058e-002f, 5.2894648e-002f, 5.2791439e-002f, 5.2688431e-002f,
5.2585624e-002f, 5.2483018e-002f, 5.2380612e-002f, 5.2278406e-002f,
5.2176400e-002f, 5.2074592e-002f, 5.1972983e-002f, 5.1871572e-002f,
5.1770360e-002f, 5.1669344e-002f, 5.1568526e-002f, 5.1467905e-002f,
5.1367480e-002f, 5.1267250e-002f, 5.1167217e-002f, 5.1067378e-002f,
5.0967735e-002f, 5.0868285e-002f, 5.0769030e-002f, 5.0669969e-002f,
5.0571101e-002f, 5.0472425e-002f, 5.0373943e-002f, 5.0275652e-002f,
5.0177553e-002f, 5.0079646e-002f, 4.9981929e-002f, 4.9884404e-002f,
4.9787068e-002f, 4.9689923e-002f, 4.9592967e-002f, 4.9496200e-002f,
4.9399622e-002f, 4.9303233e-002f, 4.9207031e-002f, 4.9111018e-002f,
4.9015191e-002f, 4.8919552e-002f, 4.8824099e-002f, 4.8728833e-002f,
4.8633752e-002f, 4.8538857e-002f, 4.8444147e-002f, 4.8349622e-002f,
4.8255281e-002f, 4.8161125e-002f, 4.8067152e-002f, 4.7973362e-002f,
4.7879756e-002f, 4.7786332e-002f, 4.7693090e-002f, 4.7600031e-002f,
4.7507153e-002f, 4.7414456e-002f, 4.7321940e-002f, 4.7229604e-002f,
4.7137449e-002f, 4.7045473e-002f, 4.6953677e-002f, 4.6862061e-002f,
4.6770622e-002f, 4.6679363e-002f, 4.6588281e-002f, 4.6497377e-002f,
4.6406651e-002f, 4.6316101e-002f, 4.6225728e-002f, 4.6135532e-002f,
4.6045511e-002f, 4.5955666e-002f, 4.5865997e-002f, 4.5776502e-002f,
4.5687182e-002f, 4.5598036e-002f, 4.5509065e-002f, 4.5420266e-002f,
4.5331642e-002f, 4.5243190e-002f, 4.5154910e-002f, 4.5066803e-002f,
4.4978868e-002f, 4.4891104e-002f, 4.4803512e-002f, 4.4716091e-002f,

4.4628840e-002f, 4.4541759e-002f, 4.4454848e-002f, 4.4368107e-002f,
4.4281535e-002f, 4.4195132e-002f, 4.4108898e-002f, 4.4022832e-002f,
4.3936934e-002f, 4.3851203e-002f, 4.3765640e-002f, 4.3680243e-002f,
4.3595014e-002f, 4.3509950e-002f, 4.3425053e-002f, 4.3340321e-002f,
4.3255755e-002f, 4.3171353e-002f, 4.3087116e-002f, 4.3003044e-002f,
4.2919136e-002f, 4.2835391e-002f, 4.2751810e-002f, 4.2668392e-002f,
4.2585136e-002f, 4.2502043e-002f, 4.2419113e-002f, 4.2336344e-002f,
4.2253736e-002f, 4.2171290e-002f, 4.2089004e-002f, 4.2006880e-002f,
4.1924915e-002f, 4.1843110e-002f, 4.1761465e-002f, 4.1679979e-002f,
4.1598653e-002f, 4.1517485e-002f, 4.1436475e-002f, 4.1355623e-002f,
4.1274929e-002f, 4.1194393e-002f, 4.1114014e-002f, 4.1033791e-002f,
4.0953725e-002f, 4.0873816e-002f, 4.0794062e-002f, 4.0714464e-002f,
4.0635021e-002f, 4.0555733e-002f, 4.0476600e-002f, 4.0397621e-002f,
4.0318797e-002f, 4.0240126e-002f, 4.0161609e-002f, 4.0083244e-002f,
4.0005033e-002f, 3.9926975e-002f, 3.9849068e-002f, 3.9771314e-002f,
3.9693712e-002f, 3.9616261e-002f, 3.9538961e-002f, 3.9461811e-002f,
3.9384813e-002f, 3.9307964e-002f, 3.9231266e-002f, 3.9154717e-002f,
3.9078318e-002f, 3.9002067e-002f, 3.8925966e-002f, 3.8850013e-002f,
3.8774208e-002f, 3.8698551e-002f, 3.8623042e-002f, 3.8547680e-002f,
3.8472465e-002f, 3.8397396e-002f, 3.8322475e-002f, 3.8247699e-002f,
3.8173069e-002f, 3.8098585e-002f, 3.8024247e-002f, 3.7950053e-002f,
3.7876004e-002f, 3.7802100e-002f, 3.7728340e-002f, 3.7654723e-002f,
3.7581251e-002f, 3.7507922e-002f, 3.7434735e-002f, 3.7361692e-002f,
3.7288791e-002f, 3.7216033e-002f, 3.7143416e-002f, 3.7070941e-002f,
3.6998608e-002f, 3.6926415e-002f, 3.6854364e-002f, 3.6782453e-002f,
3.6710682e-002f, 3.6639052e-002f, 3.6567561e-002f, 3.6496209e-002f,
3.6424997e-002f, 3.6353924e-002f, 3.6282990e-002f, 3.6212194e-002f,
3.6141536e-002f, 3.6071016e-002f, 3.6000633e-002f, 3.5930388e-002f,
3.5860280e-002f, 3.5790309e-002f, 3.5720474e-002f, 3.5650776e-002f,
3.5581213e-002f, 3.5511786e-002f, 3.5442495e-002f, 3.5373339e-002f,
3.5304318e-002f, 3.5235432e-002f, 3.5166679e-002f, 3.5098062e-002f,
3.5029578e-002f, 3.4961227e-002f, 3.4893010e-002f, 3.4824926e-002f,
3.4756975e-002f, 3.4689157e-002f, 3.4621471e-002f, 3.4553917e-002f,
3.4486494e-002f, 3.4419204e-002f, 3.4352044e-002f, 3.4285016e-002f,
3.4218118e-002f, 3.4151351e-002f, 3.4084715e-002f, 3.4018208e-002f,
3.3951831e-002f, 3.3885583e-002f, 3.3819465e-002f, 3.3753476e-002f,
3.3687616e-002f, 3.3621884e-002f, 3.3556280e-002f, 3.3490804e-002f,
3.3425456e-002f, 3.3360236e-002f, 3.3295143e-002f, 3.3230177e-002f,
3.3165337e-002f, 3.3100625e-002f, 3.3036038e-002f, 3.2971578e-002f,
3.2907243e-002f, 3.2843034e-002f, 3.2778950e-002f, 3.2714991e-002f,
3.2651157e-002f, 3.2587447e-002f, 3.2523862e-002f, 3.2460401e-002f,
3.2397063e-002f, 3.2333850e-002f, 3.2270759e-002f, 3.2207792e-002f,
3.2144947e-002f, 3.2082225e-002f, 3.2019626e-002f, 3.1957149e-002f,
3.1894793e-002f, 3.1832560e-002f, 3.1770447e-002f, 3.1708456e-002f,
3.1646586e-002f, 3.1584837e-002f, 3.1523208e-002f, 3.1461699e-002f,
3.1400310e-002f, 3.1339042e-002f, 3.1277892e-002f, 3.1216862e-002f,
3.1155951e-002f, 3.1095159e-002f, 3.1034486e-002f, 3.0973931e-002f,
3.0913494e-002f, 3.0853175e-002f, 3.0792973e-002f, 3.0732890e-002f,
3.0672923e-002f, 3.0613073e-002f, 3.0553341e-002f, 3.0493724e-002f,
3.0434224e-002f, 3.0374841e-002f, 3.0315573e-002f, 3.0256420e-002f,
3.0197383e-002f, 3.0138462e-002f, 3.0079655e-002f, 3.0020963e-002f,
2.9962386e-002f, 2.9903922e-002f, 2.9845573e-002f, 2.9787338e-002f,
2.9729216e-002f, 2.9671208e-002f, 2.9613313e-002f, 2.9555531e-002f,
2.9497862e-002f, 2.9440305e-002f, 2.9382861e-002f, 2.9325528e-002f,
2.9268308e-002f, 2.9211199e-002f, 2.9154201e-002f, 2.9097315e-002f,
2.9040540e-002f, 2.8983875e-002f, 2.8927321e-002f, 2.8870878e-002f,
2.8814545e-002f, 2.8758321e-002f, 2.8702207e-002f, 2.8646203e-002f,
2.8590308e-002f, 2.8534522e-002f, 2.8478845e-002f, 2.8423276e-002f,
2.8367816e-002f, 2.8312465e-002f, 2.8257221e-002f, 2.8202085e-002f,
2.8147056e-002f, 2.8092135e-002f, 2.8037321e-002f, 2.7982614e-002f,
2.7928014e-002f, 2.7873521e-002f, 2.7819133e-002f, 2.7764852e-002f,
2.7710677e-002f, 2.7656607e-002f, 2.7602643e-002f, 2.7548784e-002f,
2.7495030e-002f, 2.7441382e-002f, 2.7387838e-002f, 2.7334398e-002f,
2.7281062e-002f, 2.7227831e-002f, 2.7174704e-002f, 2.7121680e-002f,
2.7068760e-002f, 2.7015942e-002f, 2.6963228e-002f, 2.6910617e-002f,
2.6858109e-002f, 2.6805703e-002f, 2.6753399e-002f, 2.6701197e-002f,
2.6649097e-002f, 2.6597099e-002f, 2.6545202e-002f, 2.6493407e-002f,
2.6441712e-002f, 2.6390119e-002f, 2.6338626e-002f, 2.6287234e-002f,
2.6235941e-002f, 2.6184749e-002f, 2.6133657e-002f, 2.6082665e-002f,
2.6031772e-002f, 2.5980978e-002f, 2.5930283e-002f, 2.5879688e-002f,
2.5829191e-002f, 2.5778792e-002f, 2.5728492e-002f, 2.5678290e-002f,
2.5628186e-002f, 2.5578180e-002f, 2.5528272e-002f, 2.5478460e-002f,
2.5428746e-002f, 2.5379129e-002f, 2.5329609e-002f, 2.5280185e-002f,
2.5230858e-002f, 2.5181627e-002f, 2.5132492e-002f, 2.5083453e-002f,
2.5034510e-002f, 2.4985662e-002f, 2.4936910e-002f, 2.4888252e-002f,
2.4839690e-002f, 2.4791222e-002f, 2.4742849e-002f, 2.4694571e-002f,

```
2.4646386e-002f, 2.4598296e-002f, 2.4550299e-002f, 2.4502396e-002f,
2.4454586e-002f, 2.4406870e-002f, 2.4359247e-002f, 2.4311717e-002f,
2.4264279e-002f, 2.4216934e-002f, 2.4169682e-002f, 2.4122521e-002f,
2.4075453e-002f, 2.4028477e-002f, 2.3981592e-002f, 2.3934798e-002f,
2.3888096e-002f, 2.3841486e-002f, 2.3794966e-002f, 2.3748536e-002f,
2.3702198e-002f, 2.3655950e-002f, 2.3609792e-002f, 2.3563724e-002f,
2.3517746e-002f, 2.3471858e-002f, 2.3426059e-002f, 2.3380349e-002f,
2.3334729e-002f, 2.3289198e-002f, 2.3243756e-002f, 2.3198402e-002f,
2.3153137e-002f, 2.3107960e-002f, 2.3062871e-002f, 2.3017871e-002f,
2.2972958e-002f, 2.2928133e-002f, 2.2883395e-002f, 2.2838744e-002f,
2.2794181e-002f, 2.2749704e-002f, 2.2705315e-002f, 2.2661012e-002f,
2.2616795e-002f, 2.2572665e-002f, 2.2528621e-002f, 2.2484662e-002f,
2.2440790e-002f, 2.2397003e-002f, 2.2353302e-002f, 2.2309685e-002f,
2.2266154e-002f, 2.2222708e-002f, 2.2179347e-002f, 2.2136070e-002f,
2.2092878e-002f, 2.2049770e-002f, 2.2006746e-002f, 2.1963806e-002f,
2.1920950e-002f, 2.1878177e-002f, 2.1835488e-002f, 2.1792882e-002f,
2.1750359e-002f, 2.1707920e-002f, 2.1665563e-002f, 2.1623288e-002f,
2.1581097e-002f, 2.1538987e-002f, 2.1496960e-002f, 2.1455015e-002f,
2.1413151e-002f, 2.1371370e-002f, 2.1329669e-002f, 2.1288050e-002f,
2.1246513e-002f, 2.1205056e-002f, 2.1163681e-002f, 2.1122386e-002f,
2.1081171e-002f, 2.1040037e-002f, 2.0998983e-002f, 2.0958010e-002f,
2.0917116e-002f, 2.0876302e-002f, 2.0835568e-002f, 2.0794913e-002f,
2.0754338e-002f, 2.0713842e-002f, 2.0673424e-002f, 2.0633086e-002f,
2.0592826e-002f, 2.0552645e-002f, 2.0512543e-002f, 2.0472518e-002f,
2.0432572e-002f, 2.0392703e-002f, 2.0352913e-002f, 2.0313200e-002f,
2.0273564e-002f, 2.0234006e-002f, 2.0194525e-002f, 2.0155121e-002f,
2.0115794e-002f, 2.0076544e-002f, 2.0037370e-002f, 1.9998273e-002f,
1.9959252e-002f, 1.9920307e-002f, 1.9881438e-002f, 1.9842645e-002f,
1.9803928e-002f, 1.9765286e-002f, 1.9726719e-002f, 1.9688228e-002f,
1.9649812e-002f, 1.9611471e-002f, 1.9573205e-002f, 1.9535013e-002f,
1.9496896e-002f, 1.9458853e-002f, 1.9420885e-002f, 1.9382991e-002f,
1.9345170e-002f, 1.9307423e-002f, 1.9269750e-002f, 1.9232151e-002f,
1.9194625e-002f, 1.9157172e-002f, 1.9119792e-002f, 1.9082485e-002f,
1.9045251e-002f, 1.9008090e-002f, 1.8971001e-002f, 1.8933984e-002f,
1.8897040e-002f, 1.8860167e-002f, 1.8823367e-002f, 1.8786639e-002f,
1.8749982e-002f, 1.8713396e-002f, 1.8676882e-002f, 1.8640440e-002f,
1.8604068e-002f, 1.8567768e-002f, 1.8531538e-002f, 1.8495379e-002f,
1.8459290e-002f, 1.8423272e-002f, 1.8387324e-002f, 1.8351447e-002f,
1.8315639e-002f, 1.8279901e-002f, 1.8244233e-002f, 1.8208634e-002f,
1.8173105e-002f, 1.8137646e-002f, 1.8102255e-002f, 1.8066934e-002f,
1.8031681e-002f, 1.7996497e-002f, 1.7961382e-002f, 1.7926336e-002f,
1.7891358e-002f, 1.7856448e-002f, 1.7821606e-002f, 1.7786832e-002f,
1.7752126e-002f, 1.7717488e-002f, 1.7682917e-002f, 1.7648414e-002f,
1.7613978e-002f, 1.7579609e-002f, 1.7545307e-002f, 1.7511073e-002f,
1.7476905e-002f, 1.7442803e-002f, 1.7408769e-002f, 1.7374800e-002f,
1.7340898e-002f, 1.7307062e-002f, 1.7273293e-002f, 1.7239589e-002f,
1.7205950e-002f, 1.7172378e-002f, 1.7138871e-002f, 1.7105429e-002f,
1.7072053e-002f, 1.7038741e-002f, 1.7005495e-002f, 1.6972314e-002f,
1.6939197e-002f, 1.6906145e-002f, 1.6873157e-002f, 1.6840234e-002f,
1.6807375e-002f, 1.6774580e-002f, 1.6741849e-002f, 1.6709182e-002f,
1.6676579e-002f, 1.6644039e-002f, 1.6611563e-002f, 1.6579150e-002f,
1.6546801e-002f, 1.6514514e-002f, 1.6482291e-002f, 1.6450130e-002f,
1.6418033e-002f, 1.6385997e-002f, 1.6354025e-002f, 1.6322114e-002f,
1.6290266e-002f, 1.6258481e-002f, 1.6226757e-002f, 1.6195095e-002f,
1.6163495e-002f, 1.6131956e-002f, 1.6100479e-002f, 1.6069064e-002f,
1.6037709e-002f, 1.6006416e-002f, 1.5975184e-002f, 1.5944013e-002f,
1.5912903e-002f, 1.5881853e-002f, 1.5850864e-002f, 1.5819936e-002f,
1.5789068e-002f, 1.5758260e-002f, 1.5727512e-002f, 1.5696824e-002f,
1.5666196e-002f, 1.5635628e-002f, 1.5605119e-002f, 1.5574670e-002f,
1.5544281e-002f, 1.5513951e-002f, 1.5483679e-002f, 1.5453467e-002f,
1.5423314e-002f, 1.5393220e-002f, 1.5363184e-002f, 1.5333208e-002f,
1.5303289e-002f, 1.5273429e-002f, 1.5243627e-002f, 1.5213884e-002f,
1.5184198e-002f, 1.5154570e-002f, 1.5125000e-002f, 1.5095488e-002f,
1.5066034e-002f, 1.5036636e-002f, 1.5007297e-002f, 1.4978014e-002f,
1.4948789e-002f, 1.4919620e-002f, 1.4890509e-002f, 1.4861454e-002f,
1.4832456e-002f, 1.4803515e-002f, 1.4774630e-002f, 1.4745802e-002f,
1.4717029e-002f, 1.4688313e-002f, 1.4659653e-002f, 1.4631049e-002f,
1.4602500e-002f, 1.4574008e-002f, 1.4545571e-002f, 1.4517189e-002f,
1.4488863e-002f, 1.4460592e-002f, 1.4432376e-002f, 1.4404215e-002f,
1.4376110e-002f, 1.4348059e-002f, 1.4320063e-002f, 1.4292121e-002f,
1.4264234e-002f, 1.4236401e-002f, 1.4208623e-002f, 1.4180899e-002f,
1.4153229e-002f, 1.4125613e-002f, 1.4098051e-002f, 1.4070542e-002f,
1.4043087e-002f, 1.4015686e-002f, 1.3988339e-002f, 1.3961044e-002f,
1.3933803e-002f, 1.3906615e-002f, 1.3879481e-002f, 1.3852399e-002f,
1.3825370e-002f, 1.3798393e-002f, 1.3771470e-002f, 1.3744598e-002f,
1.3717780e-002f, 1.3691013e-002f, 1.3664299e-002f, 1.3637637e-002f,
```

```
1.3611027e-002f, 1.3584469e-002f, 1.3557963e-002f, 1.3531508e-002f,
1.3505105e-002f, 1.3478754e-002f, 1.3452454e-002f, 1.3426205e-002f,
1.3400008e-002f, 1.3373861e-002f, 1.3347766e-002f, 1.3321722e-002f,
1.3295728e-002f, 1.3269785e-002f, 1.3243893e-002f, 1.3218051e-002f,
1.3192260e-002f, 1.3166519e-002f, 1.3140828e-002f, 1.3115187e-002f,
1.3089597e-002f, 1.3064056e-002f, 1.3038565e-002f, 1.3013124e-002f,
1.2987733e-002f, 1.2962391e-002f, 1.2937098e-002f, 1.2911855e-002f,
1.2886661e-002f, 1.2861517e-002f, 1.2836421e-002f, 1.2811374e-002f,
1.2786377e-002f, 1.2761428e-002f, 1.2736527e-002f, 1.2711676e-002f,
1.2686872e-002f, 1.2662117e-002f, 1.2637411e-002f, 1.2612752e-002f,
1.2588142e-002f, 1.2563580e-002f, 1.2539066e-002f, 1.2514599e-002f,
1.2490181e-002f, 1.2465809e-002f, 1.2441486e-002f, 1.2417210e-002f,
1.2392981e-002f, 1.2368800e-002f, 1.2344666e-002f, 1.2320578e-002f,
1.2296538e-002f, 1.2272545e-002f, 1.2248599e-002f, 1.2224699e-002f,
1.2200846e-002f, 1.2177039e-002f, 1.2153279e-002f, 1.2129566e-002f,
1.2105898e-002f, 1.2082277e-002f, 1.2058702e-002f, 1.2035172e-002f,
1.2011689e-002f, 1.1988252e-002f, 1.1964860e-002f, 1.1941514e-002f,
1.1918214e-002f, 1.1894958e-002f, 1.1871749e-002f, 1.1848584e-002f,
1.1825465e-002f, 1.1802391e-002f, 1.1779362e-002f, 1.1756378e-002f,
1.1733439e-002f, 1.1710544e-002f, 1.1687694e-002f, 1.1664889e-002f,
1.1642128e-002f, 1.1619412e-002f, 1.1596740e-002f, 1.1574112e-002f,
1.1551529e-002f, 1.1528989e-002f, 1.1506494e-002f, 1.1484042e-002f,
1.1461634e-002f, 1.1439270e-002f, 1.1416949e-002f, 1.1394672e-002f,
1.1372439e-002f, 1.1350249e-002f, 1.1328102e-002f, 1.1305998e-002f,
1.1283938e-002f, 1.1261920e-002f, 1.1239946e-002f, 1.1218014e-002f,
1.1196125e-002f, 1.1174279e-002f, 1.1152476e-002f, 1.1130715e-002f,
1.1108997e-002f, 1.1087320e-002f, 1.1065687e-002f, 1.1044095e-002f,
1.1022546e-002f, 1.1001038e-002f, 1.0979573e-002f, 1.0958149e-002f,
1.0936768e-002f, 1.0915427e-002f, 1.0894129e-002f, 1.0872872e-002f,
1.0851657e-002f, 1.0830483e-002f, 1.0809350e-002f, 1.0788259e-002f,
1.0767209e-002f, 1.0746199e-002f, 1.0725231e-002f, 1.0704304e-002f,
1.0683418e-002f, 1.0662572e-002f, 1.0641767e-002f, 1.0621002e-002f,
1.0600279e-002f, 1.0579595e-002f, 1.0558952e-002f, 1.0538349e-002f,
1.0517787e-002f, 1.0497264e-002f, 1.0476782e-002f, 1.0456339e-002f,
1.0435936e-002f, 1.0415574e-002f, 1.0395251e-002f, 1.0374967e-002f,
1.0354723e-002f, 1.0334519e-002f, 1.0314354e-002f, 1.0294229e-002f,
1.0274142e-002f, 1.0254095e-002f, 1.0234087e-002f, 1.0214118e-002f,
1.0194188e-002f, 1.0174297e-002f, 1.0154445e-002f, 1.0134631e-002f,
1.0114856e-002f, 1.0095120e-002f, 1.0075422e-002f, 1.0055763e-002f,
1.0036142e-002f, 1.0016559e-002f, 9.9970148e-003f, 9.9775084e-003f,
9.9580401e-003f, 9.9386098e-003f, 9.9192174e-003f, 9.8998629e-003f,
9.8805461e-003f, 9.8612669e-003f, 9.8420255e-003f, 9.8228215e-003f,
9.8036550e-003f, 9.7845260e-003f, 9.7654342e-003f, 9.7463797e-003f,
9.7273624e-003f, 9.7083822e-003f, 9.6894390e-003f, 9.6705328e-003f,
9.6516635e-003f, 9.6328309e-003f, 9.6140352e-003f, 9.5952761e-003f,
9.5765536e-003f, 9.5578677e-003f, 9.5392182e-003f, 9.5206051e-003f,
9.5020283e-003f, 9.4834877e-003f, 9.4649834e-003f, 9.4465151e-003f,
9.4280829e-003f, 9.4096867e-003f, 9.3913263e-003f, 9.3730018e-003f,
9.3547130e-003f, 9.3364599e-003f, 9.3182424e-003f, 9.3000605e-003f,
9.2819140e-003f, 9.2638030e-003f, 9.2457273e-003f, 9.2276868e-003f,
9.2096816e-003f, 9.1917115e-003f, 9.1737765e-003f, 9.1558764e-003f,
9.1380113e-003f, 9.1201810e-003f, 9.1023856e-003f, 9.0846248e-003f,
9.0668987e-003f, 9.0492072e-003f, 9.0315502e-003f, 9.0139277e-003f,
8.9963396e-003f, 8.9787857e-003f, 8.9612662e-003f, 8.9437808e-003f,
8.9263295e-003f, 8.9089123e-003f, 8.8915290e-003f, 8.8741797e-003f,
8.8568642e-003f, 8.8395826e-003f, 8.8223346e-003f, 8.8051203e-003f,
8.7879396e-003f, 8.7707924e-003f, 8.7536786e-003f, 8.7365983e-003f,
8.7195513e-003f, 8.7025375e-003f, 8.6855570e-003f, 8.6686096e-003f,
8.6516952e-003f, 8.6348139e-003f, 8.6179654e-003f, 8.6011499e-003f,
8.5843672e-003f, 8.5676172e-003f, 8.5508999e-003f, 8.5342152e-003f,
8.5175631e-003f, 8.5009435e-003f, 8.4843563e-003f, 8.4678014e-003f,
8.4512789e-003f, 8.4347886e-003f, 8.4183305e-003f, 8.4019045e-003f,
8.3855105e-003f, 8.3691486e-003f, 8.3528185e-003f, 8.3365203e-003f,
8.3202540e-003f, 8.3040193e-003f, 8.2878164e-003f, 8.2716450e-003f,
8.2555052e-003f, 8.2393969e-003f, 8.2233201e-003f, 8.2072746e-003f,
8.1912604e-003f, 8.1752774e-003f, 8.1593257e-003f, 8.1434051e-003f,
8.1275155e-003f, 8.1116569e-003f, 8.0958293e-003f, 8.0800326e-003f,
8.0642667e-003f, 8.0485315e-003f, 8.0328271e-003f, 8.0171533e-003f,
8.0015100e-003f, 7.9858973e-003f, 7.9703151e-003f, 7.9547633e-003f,
7.9392418e-003f, 7.9237506e-003f, 7.9082896e-003f, 7.8928588e-003f,
7.8774581e-003f, 7.8620875e-003f, 7.8467468e-003f, 7.8314361e-003f,
7.8161553e-003f, 7.8009042e-003f, 7.7856830e-003f, 7.7704914e-003f,
7.7553295e-003f, 7.7401971e-003f, 7.7250943e-003f, 7.7100209e-003f,
7.6949770e-003f, 7.6799624e-003f, 7.6649771e-003f, 7.6500211e-003f,
7.6350942e-003f, 7.6201965e-003f, 7.6053278e-003f, 7.5904881e-003f,
7.5756774e-003f, 7.5608956e-003f, 7.5461427e-003f, 7.5314185e-003f,
```

```
7.5167231e-003f, 7.5020563e-003f, 7.4874181e-003f, 7.4728085e-003f,
7.4582275e-003f, 7.4436748e-003f, 7.4291506e-003f, 7.4146547e-003f,
7.4001871e-003f, 7.3857477e-003f, 7.3713365e-003f, 7.3569534e-003f,
7.3425984e-003f, 7.3282713e-003f, 7.3139723e-003f, 7.2997011e-003f,
7.2854578e-003f, 7.2712423e-003f, 7.2570545e-003f, 7.2428944e-003f,
7.2287619e-003f, 7.2146570e-003f, 7.2005796e-003f, 7.1865297e-003f,
7.1725072e-003f, 7.1585121e-003f, 7.1445443e-003f, 7.1306037e-003f,
7.1166904e-003f, 7.1028041e-003f, 7.0889450e-003f, 7.0751129e-003f,
7.0613078e-003f, 7.0475297e-003f, 7.0337784e-003f, 7.0200540e-003f,
7.0063563e-003f, 6.9926854e-003f, 6.9790411e-003f, 6.9654235e-003f,
6.9518324e-003f, 6.9382678e-003f, 6.9247298e-003f, 6.9112181e-003f,
6.8977328e-003f, 6.8842738e-003f, 6.8708411e-003f, 6.8574346e-003f,
6.8440542e-003f, 6.8307000e-003f, 6.8173718e-003f, 6.8040696e-003f,
6.7907934e-003f, 6.7775430e-003f, 6.7643186e-003f, 6.7511199e-003f,
6.7379470e-003f, 6.7247998e-003f, 6.7116782e-003f, 6.6985823e-003f,
6.6855119e-003f, 6.6724670e-003f, 6.6594475e-003f, 6.6464535e-003f,
6.6334848e-003f, 6.6205414e-003f, 6.6076233e-003f, 6.5947304e-003f,
6.5818626e-003f, 6.5690200e-003f, 6.5562024e-003f, 6.5434098e-003f,
6.5306422e-003f, 6.5178995e-003f, 6.5051816e-003f, 6.4924886e-003f,
6.4798203e-003f, 6.4671768e-003f, 6.4545579e-003f, 6.4419636e-003f,
6.4293939e-003f, 6.4168488e-003f, 6.4043281e-003f, 6.3918319e-003f,
6.3793600e-003f, 6.3669125e-003f, 6.3544892e-003f, 6.3420902e-003f,
6.3297154e-003f, 6.3173648e-003f, 6.3050382e-003f, 6.2927357e-003f,
6.2804572e-003f, 6.2682026e-003f, 6.2559720e-003f, 6.2437652e-003f,
6.2315823e-003f, 6.2194231e-003f, 6.2072876e-003f, 6.1951759e-003f,
6.1830877e-003f, 6.1710232e-003f, 6.1589822e-003f, 6.1469646e-003f,
6.1349706e-003f, 6.1229999e-003f, 6.1110526e-003f, 6.0991286e-003f,
6.0872278e-003f, 6.0753503e-003f, 6.0634960e-003f, 6.0516648e-003f,
6.0398567e-003f, 6.0280716e-003f, 6.0163095e-003f, 6.0045703e-003f,
5.9928541e-003f, 5.9811607e-003f, 5.9694902e-003f, 5.9578424e-003f,
5.9462174e-003f, 5.9346150e-003f, 5.9230353e-003f, 5.9114781e-003f,
5.8999435e-003f, 5.8884314e-003f, 5.8769418e-003f, 5.8654746e-003f,
5.8540298e-003f, 5.8426073e-003f, 5.8312071e-003f, 5.8198291e-003f,
5.8084734e-003f, 5.7971398e-003f, 5.7858283e-003f, 5.7745389e-003f,
5.7632715e-003f, 5.7520261e-003f, 5.7408026e-003f, 5.7296011e-003f,
5.7184213e-003f, 5.7072635e-003f, 5.6961273e-003f, 5.6850129e-003f,
5.6739202e-003f, 5.6628492e-003f, 5.6517997e-003f, 5.6407718e-003f,
5.6297654e-003f, 5.6187805e-003f, 5.6078171e-003f, 5.5968750e-003f,
5.5859543e-003f, 5.5750548e-003f, 5.5641767e-003f, 5.5533198e-003f,
5.5424840e-003f, 5.5316694e-003f, 5.5208759e-003f, 5.5101035e-003f,
5.4993521e-003f, 5.4886216e-003f, 5.4779121e-003f, 5.4672235e-003f,
5.4565558e-003f, 5.4459088e-003f, 5.4352827e-003f, 5.4246772e-003f,
5.4140925e-003f, 5.4035284e-003f, 5.3929850e-003f, 5.3824621e-003f,
5.3719597e-003f, 5.3614778e-003f, 5.3510164e-003f, 5.3405754e-003f,
5.3301548e-003f, 5.3197545e-003f, 5.3093745e-003f, 5.2990147e-003f,
5.2886752e-003f, 5.2783558e-003f, 5.2680566e-003f, 5.2577775e-003f,
5.2475184e-003f, 5.2372793e-003f, 5.2270603e-003f, 5.2168611e-003f,
5.2066819e-003f, 5.1965225e-003f, 5.1863830e-003f, 5.1762632e-003f,
5.1661632e-003f, 5.1560829e-003f, 5.1460222e-003f, 5.1359812e-003f,
5.1259598e-003f, 5.1159579e-003f, 5.1059755e-003f, 5.0960127e-003f,
5.0860692e-003f, 5.0761452e-003f, 5.0662405e-003f, 5.0563552e-003f,
5.0464891e-003f, 5.0366423e-003f, 5.0268147e-003f, 5.0170063e-003f,
5.0072170e-003f, 4.9974469e-003f, 4.9876957e-003f, 4.9779637e-003f,
4.9682506e-003f, 4.9585564e-003f, 4.9488812e-003f, 4.9392248e-003f,
4.9295873e-003f, 4.9199686e-003f, 4.9103687e-003f, 4.9007875e-003f,
4.8912250e-003f, 4.8816811e-003f, 4.8721559e-003f, 4.8626493e-003f,
4.8531612e-003f, 4.8436916e-003f, 4.8342405e-003f, 4.8248078e-003f,
4.8153936e-003f, 4.8059977e-003f, 4.7966201e-003f, 4.7872609e-003f,
4.7779199e-003f, 4.7685971e-003f, 4.7592925e-003f, 4.7500061e-003f,
4.7407378e-003f, 4.7314876e-003f, 4.7222554e-003f, 4.7130413e-003f,
4.7038451e-003f, 4.6946669e-003f, 4.6855065e-003f, 4.6763641e-003f,
4.6672395e-003f, 4.6581327e-003f, 4.6490436e-003f, 4.6399723e-003f,
4.6309187e-003f, 4.6218828e-003f, 4.6128645e-003f, 4.6038638e-003f,
4.5948806e-003f, 4.5859150e-003f, 4.5769669e-003f, 4.5680362e-003f,
4.5591230e-003f, 4.5502271e-003f, 4.5413487e-003f, 4.5324875e-003f,
4.5236436e-003f, 4.5148170e-003f, 4.5060076e-003f, 4.4972154e-003f,
4.4884403e-003f, 4.4796824e-003f, 4.4709416e-003f, 4.4622178e-003f,
4.4535110e-003f, 4.4448212e-003f, 4.4361484e-003f, 4.4274925e-003f,
4.4188535e-003f, 4.4102314e-003f, 4.4016260e-003f, 4.3930375e-003f,
4.3844657e-003f, 4.3759107e-003f, 4.3673723e-003f, 4.3588506e-003f,
4.3503456e-003f, 4.3418571e-003f, 4.3333852e-003f, 4.3249298e-003f,
4.3164909e-003f, 4.3080685e-003f, 4.2996625e-003f, 4.2912729e-003f,
4.2828997e-003f, 4.2745428e-003f, 4.2662023e-003f, 4.2578780e-003f,
4.2495699e-003f, 4.2412781e-003f, 4.2330024e-003f, 4.2247429e-003f,
4.2164995e-003f, 4.2082722e-003f, 4.2000609e-003f, 4.1918657e-003f,
4.1836864e-003f, 4.1755231e-003f, 4.1673758e-003f, 4.1592443e-003f,
```

```
4.1511287e-003f, 4.1430290e-003f, 4.1349450e-003f, 4.1268768e-003f,
4.1188244e-003f, 4.1107877e-003f, 4.1027666e-003f, 4.0947612e-003f,
4.0867714e-003f, 4.0787973e-003f, 4.0708386e-003f, 4.0628955e-003f,
4.0549679e-003f, 4.0470558e-003f, 4.0391591e-003f, 4.0312778e-003f,
4.0234119e-003f, 4.0155614e-003f, 4.0077261e-003f, 3.9999062e-003f,
3.9921015e-003f, 3.9843120e-003f, 3.9765378e-003f, 3.9687787e-003f,
3.9610347e-003f, 3.9533059e-003f, 3.9455921e-003f, 3.9378934e-003f,
3.9302097e-003f, 3.9225410e-003f, 3.9148872e-003f, 3.9072484e-003f,
3.8996245e-003f, 3.8920155e-003f, 3.8844214e-003f, 3.8768420e-003f,
3.8692774e-003f, 3.8617276e-003f, 3.8541925e-003f, 3.8466722e-003f,
3.8391665e-003f, 3.8316754e-003f, 3.8241990e-003f, 3.8167371e-003f,
3.8092898e-003f, 3.8018571e-003f, 3.7944388e-003f, 3.7870350e-003f,
3.7796457e-003f, 3.7722708e-003f, 3.7649103e-003f, 3.7575641e-003f,
3.7502323e-003f, 3.7429148e-003f, 3.7356115e-003f, 3.7283225e-003f,
3.7210477e-003f, 3.7137872e-003f, 3.7065407e-003f, 3.6993085e-003f,
3.6920903e-003f, 3.6848862e-003f, 3.6776962e-003f, 3.6705202e-003f,
3.6633582e-003f, 3.6562102e-003f, 3.6490762e-003f, 3.6419560e-003f,
3.6348498e-003f, 3.6277574e-003f, 3.6206788e-003f, 3.6136141e-003f,
3.6065631e-003f, 3.5995259e-003f, 3.5925025e-003f, 3.5854927e-003f,
3.5784966e-003f, 3.5715142e-003f, 3.5645454e-003f, 3.5575902e-003f,
3.5506486e-003f, 3.5437205e-003f, 3.5368059e-003f, 3.5299048e-003f,
3.5230172e-003f, 3.5161430e-003f, 3.5092822e-003f, 3.5024349e-003f,
3.4956009e-003f, 3.4887802e-003f, 3.4819728e-003f, 3.4751787e-003f,
3.4683979e-003f, 3.4616303e-003f, 3.4548759e-003f, 3.4481347e-003f,
3.4414066e-003f, 3.4346916e-003f, 3.4279898e-003f, 3.4213011e-003f,
3.4146253e-003f, 3.4079627e-003f, 3.4013130e-003f, 3.3946763e-003f,
3.3880525e-003f, 3.3814417e-003f, 3.3748438e-003f, 3.3682587e-003f,
3.3616865e-003f, 3.3551271e-003f, 3.3485805e-003f, 3.3420467e-003f,
3.3355256e-003f, 3.3290173e-003f, 3.3225217e-003f, 3.3160387e-003f,
3.3095684e-003f, 3.3031107e-003f, 3.2966656e-003f, 3.2902331e-003f,
3.2838131e-003f, 3.2774057e-003f, 3.2710107e-003f, 3.2646283e-003f,
3.2582583e-003f, 3.2519007e-003f, 3.2455555e-003f, 3.2392227e-003f,
3.2329023e-003f, 3.2265942e-003f, 3.2202984e-003f, 3.2140149e-003f,
3.2077437e-003f, 3.2014846e-003f, 3.1952378e-003f, 3.1890032e-003f,
3.1827808e-003f, 3.1765705e-003f, 3.1703723e-003f, 3.1641862e-003f,
3.1580122e-003f, 3.1518502e-003f, 3.1457003e-003f, 3.1395623e-003f,
3.1334364e-003f, 3.1273223e-003f, 3.1212202e-003f, 3.1151301e-003f,
3.1090518e-003f, 3.1029853e-003f, 3.0969307e-003f, 3.0908879e-003f,
3.0848569e-003f, 3.0788377e-003f, 3.0728302e-003f, 3.0668344e-003f,
3.0608504e-003f, 3.0548780e-003f, 3.0489173e-003f, 3.0429681e-003f,
3.0370307e-003f, 3.0311047e-003f, 3.0251904e-003f, 3.0192876e-003f,
3.0133963e-003f, 3.0075165e-003f, 3.0016482e-003f, 2.9957913e-003f,
2.9899459e-003f, 2.9841118e-003f, 2.9782892e-003f, 2.9724779e-003f,
2.9666779e-003f, 2.9608893e-003f, 2.9551119e-003f, 2.9493459e-003f,
2.9435910e-003f, 2.9378475e-003f, 2.9321151e-003f, 2.9263939e-003f,
2.9206838e-003f, 2.9149849e-003f, 2.9092972e-003f, 2.9036205e-003f,
2.8979549e-003f, 2.8923003e-003f, 2.8866568e-003f, 2.8810243e-003f,
2.8754028e-003f, 2.8697923e-003f, 2.8641927e-003f, 2.8586040e-003f,
2.8530263e-003f, 2.8474594e-003f, 2.8419034e-003f, 2.8363582e-003f,
2.8308238e-003f, 2.8253003e-003f, 2.8197875e-003f, 2.8142855e-003f,
2.8087942e-003f, 2.8033136e-003f, 2.7978437e-003f, 2.7923845e-003f,
2.7869360e-003f, 2.7814981e-003f, 2.7760708e-003f, 2.7706540e-003f,
2.7652479e-003f, 2.7598523e-003f, 2.7544672e-003f, 2.7490926e-003f,
2.7437285e-003f, 2.7383749e-003f, 2.7330318e-003f, 2.7276990e-003f,
2.7223767e-003f, 2.7170647e-003f, 2.7117631e-003f, 2.7064719e-003f,
2.7011910e-003f, 2.6959204e-003f, 2.6906600e-003f, 2.6854100e-003f,
2.6801701e-003f, 2.6749405e-003f, 2.6697212e-003f, 2.6645119e-003f,
2.6593129e-003f, 2.6541240e-003f, 2.6489452e-003f, 2.6437765e-003f,
2.6386180e-003f, 2.6334694e-003f, 2.6283310e-003f, 2.6232025e-003f,
2.6180841e-003f, 2.6129756e-003f, 2.6078771e-003f, 2.6027886e-003f,
2.5977100e-003f, 2.5926413e-003f, 2.5875825e-003f, 2.5825335e-003f,
2.5774944e-003f, 2.5724652e-003f, 2.5674457e-003f, 2.5624361e-003f,
2.5574362e-003f, 2.5524461e-003f, 2.5474657e-003f, 2.5424951e-003f,
2.5375341e-003f, 2.5325828e-003f, 2.5276412e-003f, 2.5227092e-003f,
2.5177868e-003f, 2.5128741e-003f, 2.5079709e-003f, 2.5030773e-003f,
2.4981933e-003f, 2.4933187e-003f, 2.4884537e-003f, 2.4835982e-003f,
2.4787522e-003f, 2.4739156e-003f, 2.4690884e-003f, 2.4642707e-003f,
2.4594624e-003f, 2.4546634e-003f, 2.4498738e-003f, 2.4450936e-003f,
2.4403227e-003f, 2.4355611e-003f, 2.4308088e-003f, 2.4260657e-003f,
2.4213319e-003f, 2.4166074e-003f, 2.4118921e-003f, 2.4071859e-003f,
2.4024890e-003f, 2.3978012e-003f, 2.3931226e-003f, 2.3884531e-003f,
2.3837927e-003f, 2.3791414e-003f, 2.3744991e-003f, 2.3698660e-003f,
2.3652419e-003f, 2.3606267e-003f, 2.3560206e-003f, 2.3514235e-003f,
2.3468354e-003f, 2.3422562e-003f, 2.3376859e-003f, 2.3331246e-003f,
2.3285722e-003f, 2.3240286e-003f, 2.3194939e-003f, 2.3149681e-003f,
2.3104511e-003f, 2.3059429e-003f, 2.3014435e-003f, 2.2969529e-003f,
```

```
2.2924710e-003f, 2.2879979e-003f, 2.2835335e-003f, 2.2790778e-003f,
2.2746309e-003f, 2.2701926e-003f, 2.2657629e-003f, 2.2613419e-003f,
2.2569295e-003f, 2.2525258e-003f, 2.2481306e-003f, 2.2437440e-003f,
2.2393660e-003f, 2.2349965e-003f, 2.2306355e-003f, 2.2262831e-003f,
2.2219391e-003f, 2.2176036e-003f, 2.2132766e-003f, 2.2089580e-003f,
2.2046478e-003f, 2.2003461e-003f, 2.1960527e-003f, 2.1917677e-003f,
2.1874911e-003f, 2.1832228e-003f, 2.1789629e-003f, 2.1747113e-003f,
2.1704679e-003f, 2.1662329e-003f, 2.1620061e-003f, 2.1577875e-003f,
2.1535772e-003f, 2.1493751e-003f, 2.1451812e-003f, 2.1409955e-003f,
2.1368179e-003f, 2.1326485e-003f, 2.1284873e-003f, 2.1243341e-003f,
2.1201891e-003f, 2.1160521e-003f, 2.1119233e-003f, 2.1078024e-003f,
2.1036896e-003f, 2.0995849e-003f, 2.0954881e-003f, 2.0913994e-003f,
2.0873186e-003f, 2.0832458e-003f, 2.0791809e-003f, 2.0751240e-003f,
2.0710750e-003f, 2.0670338e-003f, 2.0630006e-003f, 2.0589752e-003f,
2.0549577e-003f, 2.0509481e-003f, 2.0469462e-003f, 2.0429522e-003f,
2.0389659e-003f, 2.0349875e-003f, 2.0310167e-003f, 2.0270538e-003f,
2.0230986e-003f, 2.0191511e-003f, 2.0152113e-003f, 2.0112791e-003f,
2.0073547e-003f, 2.0034379e-003f, 1.9995288e-003f, 1.9956272e-003f,
1.9917333e-003f, 1.9878470e-003f, 1.9839683e-003f, 1.9800971e-003f,
1.9762335e-003f, 1.9723775e-003f, 1.9685289e-003f, 1.9646879e-003f,
1.9608544e-003f, 1.9570283e-003f, 1.9532097e-003f, 1.9493986e-003f,
1.9455949e-003f, 1.9417986e-003f, 1.9380097e-003f, 1.9342282e-003f,
1.9304541e-003f, 1.9266874e-003f, 1.9229280e-003f, 1.9191760e-003f,
1.9154312e-003f, 1.9116938e-003f, 1.9079637e-003f, 1.9042408e-003f,
1.9005252e-003f, 1.8968169e-003f, 1.8931158e-003f, 1.8894219e-003f,
1.8857352e-003f, 1.8820557e-003f, 1.8783834e-003f, 1.8747183e-003f,
1.8710603e-003f, 1.8674095e-003f, 1.8637657e-003f, 1.8601291e-003f,
1.8564996e-003f, 1.8528772e-003f, 1.8492618e-003f, 1.8456535e-003f,
1.8420522e-003f, 1.8384580e-003f, 1.8348707e-003f, 1.8312905e-003f,
1.8277172e-003f, 1.8241510e-003f, 1.8205916e-003f, 1.8170393e-003f,
1.8134938e-003f, 1.8099553e-003f, 1.8064237e-003f, 1.8028990e-003f,
1.7993811e-003f, 1.7958701e-003f, 1.7923660e-003f, 1.7888687e-003f,
1.7853782e-003f, 1.7818946e-003f, 1.7784177e-003f, 1.7749476e-003f,
1.7714843e-003f, 1.7680277e-003f, 1.7645779e-003f, 1.7611349e-003f,
1.7576985e-003f, 1.7542688e-003f, 1.7508459e-003f, 1.7474296e-003f,
1.7440200e-003f, 1.7406170e-003f, 1.7372207e-003f, 1.7338310e-003f,
1.7304479e-003f, 1.7270714e-003f, 1.7237015e-003f, 1.7203382e-003f,
1.7169815e-003f, 1.7136312e-003f, 1.7102876e-003f, 1.7069504e-003f,
1.7036198e-003f, 1.7002957e-003f, 1.6969780e-003f, 1.6936668e-003f,
1.6903621e-003f, 1.6870639e-003f, 1.6837720e-003f, 1.6804866e-003f,
1.6772076e-003f, 1.6739350e-003f, 1.6706688e-003f, 1.6674090e-003f,
1.6641555e-003f, 1.6609084e-003f, 1.6576676e-003f, 1.6544331e-003f,
1.6512049e-003f, 1.6479831e-003f, 1.6447675e-003f, 1.6415582e-003f,
1.6383551e-003f, 1.6351584e-003f, 1.6319678e-003f, 1.6287835e-003f,
1.6256054e-003f, 1.6224335e-003f, 1.6192677e-003f, 1.6161082e-003f,
1.6129548e-003f, 1.6098076e-003f, 1.6066665e-003f, 1.6035315e-003f,
1.6004027e-003f, 1.5972800e-003f, 1.5941633e-003f, 1.5910527e-003f,
1.5879483e-003f, 1.5848498e-003f, 1.5817574e-003f, 1.5786711e-003f,
1.5755907e-003f, 1.5725164e-003f, 1.5694481e-003f, 1.5663858e-003f,
1.5633294e-003f, 1.5602790e-003f, 1.5572346e-003f, 1.5541961e-003f,
1.5511635e-003f, 1.5481368e-003f, 1.5451161e-003f, 1.5421012e-003f,
1.5390922e-003f, 1.5360891e-003f, 1.5330919e-003f, 1.5301005e-003f,
1.5271149e-003f, 1.5241352e-003f, 1.5211613e-003f, 1.5181931e-003f,
1.5152308e-003f, 1.5122743e-003f, 1.5093235e-003f, 1.5063785e-003f,
1.5034392e-003f, 1.5005057e-003f, 1.4975778e-003f, 1.4946557e-003f,
1.4917393e-003f, 1.4888286e-003f, 1.4859236e-003f, 1.4830242e-003f,
1.4801305e-003f, 1.4772425e-003f, 1.4743600e-003f, 1.4714832e-003f,
1.4686121e-003f, 1.4657465e-003f, 1.4628865e-003f, 1.4600321e-003f,
1.4571832e-003f, 1.4543399e-003f, 1.4515022e-003f, 1.4486700e-003f,
1.4458433e-003f, 1.4430222e-003f, 1.4402065e-003f, 1.4373964e-003f,
1.4345917e-003f, 1.4317925e-003f, 1.4289988e-003f, 1.4262105e-003f,
1.4234276e-003f, 1.4206502e-003f, 1.4178782e-003f, 1.4151116e-003f,
1.4123504e-003f, 1.4095946e-003f, 1.4068442e-003f, 1.4040991e-003f,
1.4013594e-003f, 1.3986251e-003f, 1.3958960e-003f, 1.3931723e-003f,
1.3904540e-003f, 1.3877409e-003f, 1.3850331e-003f, 1.3823306e-003f,
1.3796334e-003f, 1.3769414e-003f, 1.3742547e-003f, 1.3715732e-003f,
1.3688970e-003f, 1.3662259e-003f, 1.3635601e-003f, 1.3608995e-003f,
1.3582441e-003f, 1.3555939e-003f, 1.3529488e-003f, 1.3503089e-003f,
1.3476742e-003f, 1.3450446e-003f, 1.3424201e-003f, 1.3398007e-003f,
1.3371865e-003f, 1.3345774e-003f, 1.3319733e-003f, 1.3293743e-003f,
1.3267804e-003f, 1.3241916e-003f, 1.3216078e-003f, 1.3190291e-003f,
1.3164553e-003f, 1.3138867e-003f, 1.3113230e-003f, 1.3087643e-003f,
1.3062106e-003f, 1.3036619e-003f, 1.3011182e-003f, 1.2985794e-003f,
1.2960456e-003f, 1.2935167e-003f, 1.2909928e-003f, 1.2884738e-003f,
1.2859597e-003f, 1.2834505e-003f, 1.2809462e-003f, 1.2784468e-003f,
1.2759523e-003f, 1.2734626e-003f, 1.2709778e-003f, 1.2684978e-003f,
```

1.2660227e-003f, 1.2635524e-003f, 1.2610870e-003f, 1.2586263e-003f,
1.2561705e-003f, 1.2537194e-003f, 1.2512731e-003f, 1.2488316e-003f,
1.2463949e-003f, 1.2439629e-003f, 1.2415356e-003f, 1.2391131e-003f,
1.2366953e-003f, 1.2342823e-003f, 1.2318739e-003f, 1.2294703e-003f,
1.2270713e-003f, 1.2246770e-003f, 1.2222874e-003f, 1.2199025e-003f,
1.2175222e-003f, 1.2151465e-003f, 1.2127755e-003f, 1.2104091e-003f,
1.2080473e-003f, 1.2056902e-003f, 1.2033376e-003f, 1.2009896e-003f,
1.1986462e-003f, 1.1963074e-003f, 1.1939732e-003f, 1.1916434e-003f,
1.1893183e-003f, 1.1869977e-003f, 1.1846816e-003f, 1.1823700e-003f,
1.1800629e-003f, 1.1777604e-003f, 1.1754623e-003f, 1.1731687e-003f,
1.1708796e-003f, 1.1685950e-003f, 1.1663148e-003f, 1.1640391e-003f,
1.1617678e-003f, 1.1595009e-003f, 1.1572385e-003f, 1.1549804e-003f,
1.1527268e-003f, 1.1504776e-003f, 1.1482328e-003f, 1.1459923e-003f,
1.1437562e-003f, 1.1415245e-003f, 1.1392971e-003f, 1.1370741e-003f,
1.1348554e-003f, 1.1326411e-003f, 1.1304311e-003f, 1.1282253e-003f,
1.1260239e-003f, 1.1238268e-003f, 1.1216340e-003f, 1.1194454e-003f,
1.1172611e-003f, 1.1150811e-003f, 1.1129054e-003f, 1.1107338e-003f,
1.1085665e-003f, 1.1064035e-003f, 1.1042447e-003f, 1.1020900e-003f,
1.0999396e-003f, 1.0977934e-003f, 1.0956514e-003f, 1.0935135e-003f,
1.0913798e-003f, 1.0892503e-003f, 1.0871249e-003f, 1.0850037e-003f,
1.0828866e-003f, 1.0807737e-003f, 1.0786649e-003f, 1.0765601e-003f,
1.0744595e-003f, 1.0723630e-003f, 1.0702706e-003f, 1.0681823e-003f,
1.0660980e-003f, 1.0640178e-003f, 1.0619417e-003f, 1.0598696e-003f,
1.0578016e-003f, 1.0557376e-003f, 1.0536776e-003f, 1.0516217e-003f,
1.0495697e-003f, 1.0475218e-003f, 1.0454778e-003f, 1.0434379e-003f,
1.0414019e-003f, 1.0393699e-003f, 1.0373419e-003f, 1.0353178e-003f,
1.0332976e-003f, 1.0312814e-003f, 1.0292692e-003f, 1.0272609e-003f,
1.0252565e-003f, 1.0232560e-003f, 1.0212594e-003f, 1.0192667e-003f,
1.0172778e-003f, 1.0152929e-003f, 1.0133119e-003f, 1.0113347e-003f,
1.0093613e-003f, 1.0073918e-003f, 1.0054262e-003f, 1.0034644e-003f,
1.0015064e-003f, 9.9955225e-004f, 9.9760191e-004f, 9.9565537e-004f,
9.9371263e-004f, 9.9177368e-004f, 9.8983851e-004f, 9.8790712e-004f,
9.8597949e-004f, 9.8405563e-004f, 9.8213552e-004f, 9.8021916e-004f,
9.7830654e-004f, 9.7639765e-004f, 9.7449249e-004f, 9.7259104e-004f,
9.7069330e-004f, 9.6879926e-004f, 9.6690892e-004f, 9.6502227e-004f,
9.6313930e-004f, 9.6126001e-004f, 9.5938438e-004f, 9.5751241e-004f,
9.5564409e-004f, 9.5377942e-004f, 9.5191839e-004f, 9.5006099e-004f,
9.4820721e-004f, 9.4635705e-004f, 9.4451050e-004f, 9.4266756e-004f,
9.4082821e-004f, 9.3899244e-004f, 9.3716026e-004f, 9.3533166e-004f,
9.3350662e-004f, 9.3168515e-004f, 9.2986723e-004f, 9.2805285e-004f,
9.2624202e-004f, 9.2443472e-004f, 9.2263094e-004f, 9.2083069e-004f,
9.1903394e-004f, 9.1724071e-004f, 9.1545097e-004f, 9.1366473e-004f,
9.1188197e-004f, 9.1010268e-004f, 9.0832687e-004f, 9.0655453e-004f,
9.0478564e-004f, 9.0302021e-004f, 9.0125822e-004f, 8.9949967e-004f,
8.9774455e-004f, 8.9599285e-004f, 8.9424457e-004f, 8.9249970e-004f,
8.9075824e-004f, 8.8902018e-004f, 8.8728551e-004f, 8.8555422e-004f,
8.8382631e-004f, 8.8210177e-004f, 8.8038059e-004f, 8.7866278e-004f,
8.7694832e-004f, 8.7523720e-004f, 8.7352942e-004f, 8.7182497e-004f,
8.7012385e-004f, 8.6842605e-004f, 8.6673156e-004f, 8.6504038e-004f,
8.6335249e-004f, 8.6166790e-004f, 8.5998660e-004f, 8.5830858e-004f,
8.5663383e-004f, 8.5496235e-004f, 8.5329413e-004f, 8.5162917e-004f,
8.4996745e-004f, 8.4830898e-004f, 8.4665374e-004f, 8.4500174e-004f,
8.4335295e-004f, 8.4170739e-004f, 8.4006503e-004f, 8.3842588e-004f,
8.3678993e-004f, 8.3515717e-004f, 8.3352759e-004f, 8.3190120e-004f,
8.3027798e-004f, 8.2865792e-004f, 8.2704103e-004f, 8.2542729e-004f,
8.2381670e-004f, 8.2220926e-004f, 8.2060495e-004f, 8.1900377e-004f,
8.1740571e-004f, 8.1581077e-004f, 8.1421895e-004f, 8.1263023e-004f,
8.1104461e-004f, 8.0946208e-004f, 8.0788265e-004f, 8.0630629e-004f,
8.0473301e-004f, 8.0316280e-004f, 8.0159565e-004f, 8.0003156e-004f,
7.9847053e-004f, 7.9691254e-004f, 7.9535759e-004f, 7.9380567e-004f,
7.9225678e-004f, 7.9071091e-004f, 7.8916806e-004f, 7.8762823e-004f,
7.8609139e-004f, 7.8455755e-004f, 7.8302671e-004f, 7.8149885e-004f,
7.7997398e-004f, 7.7845208e-004f, 7.7693315e-004f, 7.7541718e-004f,
7.7390417e-004f, 7.7239412e-004f, 7.7088701e-004f, 7.6938284e-004f,
7.6788160e-004f, 7.6638330e-004f, 7.6488792e-004f, 7.6339545e-004f,
7.6190590e-004f, 7.6041926e-004f, 7.5893551e-004f, 7.5745466e-004f,
7.5597670e-004f, 7.5450163e-004f, 7.5302943e-004f, 7.5156010e-004f,
7.5009364e-004f, 7.4863005e-004f, 7.4716931e-004f, 7.4571142e-004f,
7.4425637e-004f, 7.4280416e-004f, 7.4135479e-004f, 7.3990824e-004f,
7.3846452e-004f, 7.3702361e-004f, 7.3558552e-004f, 7.3415023e-004f,
7.3271774e-004f, 7.3128805e-004f, 7.2986115e-004f, 7.2843703e-004f,
7.2701569e-004f, 7.2559712e-004f, 7.2418132e-004f, 7.2276829e-004f,
7.2135801e-004f, 7.1995048e-004f, 7.1854570e-004f, 7.1714366e-004f,
7.1574436e-004f, 7.1434778e-004f, 7.1295393e-004f, 7.1156280e-004f,
7.1017439e-004f, 7.0878868e-004f, 7.0740568e-004f, 7.0602538e-004f,
7.0464777e-004f, 7.0327285e-004f, 7.0190061e-004f, 7.0053104e-004f,

6.9916416e-004f, 6.9779993e-004f, 6.9643837e-004f, 6.9507947e-004f,
6.9372322e-004f, 6.9236961e-004f, 6.9101865e-004f, 6.8967032e-004f,
6.8832462e-004f, 6.8698155e-004f, 6.8564110e-004f, 6.8430326e-004f,
6.8296804e-004f, 6.8163542e-004f, 6.8030540e-004f, 6.7897797e-004f,
6.7765314e-004f, 6.7633089e-004f, 6.7501122e-004f, 6.7369412e-004f,
6.7237960e-004f, 6.7106764e-004f, 6.6975824e-004f, 6.6845139e-004f,
6.6714710e-004f, 6.6584535e-004f, 6.6454614e-004f, 6.6324946e-004f,
6.6195532e-004f, 6.6066370e-004f, 6.5937460e-004f, 6.5808801e-004f,
6.5680394e-004f, 6.5552237e-004f, 6.5424330e-004f, 6.5296673e-004f,
6.5169265e-004f, 6.5042106e-004f, 6.4915194e-004f, 6.4788531e-004f,
6.4662114e-004f, 6.4535944e-004f, 6.4410020e-004f, 6.4284342e-004f,
6.4158909e-004f, 6.4033721e-004f, 6.3908777e-004f, 6.3784077e-004f,
6.3659621e-004f, 6.3535407e-004f, 6.3411435e-004f, 6.3287706e-004f,
6.3164218e-004f, 6.3040970e-004f, 6.2917964e-004f, 6.2795197e-004f,
6.2672670e-004f, 6.2550382e-004f, 6.2428332e-004f, 6.2306521e-004f,
6.2184947e-004f, 6.2063611e-004f, 6.1942511e-004f, 6.1821648e-004f,
6.1701020e-004f, 6.1580628e-004f, 6.1460471e-004f, 6.1340548e-004f,
6.1220859e-004f, 6.1101404e-004f, 6.0982182e-004f, 6.0863192e-004f,
6.0744435e-004f, 6.0625909e-004f, 6.0507614e-004f, 6.0389551e-004f,
6.0271718e-004f, 6.0154114e-004f, 6.0036740e-004f, 5.9919596e-004f,
5.9802679e-004f, 5.9685991e-004f, 5.9569531e-004f, 5.9453298e-004f,
5.9337291e-004f, 5.9221511e-004f, 5.9105957e-004f, 5.8990628e-004f,
5.8875525e-004f, 5.8760646e-004f, 5.8645991e-004f, 5.8531560e-004f,
5.8417352e-004f, 5.8303367e-004f, 5.8189604e-004f, 5.8076063e-004f,
5.7962744e-004f, 5.7849646e-004f, 5.7736769e-004f, 5.7624112e-004f,
5.7511675e-004f, 5.7399457e-004f, 5.7287458e-004f, 5.7175678e-004f,
5.7064115e-004f, 5.6952771e-004f, 5.6841643e-004f, 5.6730733e-004f,
5.6620039e-004f, 5.6509561e-004f, 5.6399298e-004f, 5.6289251e-004f,
5.6179418e-004f, 5.6069800e-004f, 5.5960395e-004f, 5.5851204e-004f,
5.5742226e-004f, 5.5633461e-004f, 5.5524908e-004f, 5.5416567e-004f,
5.5308437e-004f, 5.5200518e-004f, 5.5092810e-004f, 5.4985312e-004f,
5.4878023e-004f, 5.4770944e-004f, 5.4664074e-004f, 5.4557413e-004f,
5.4450959e-004f, 5.4344713e-004f, 5.4238675e-004f, 5.4132843e-004f,
5.4027218e-004f, 5.3921800e-004f, 5.3816586e-004f, 5.3711578e-004f,
5.3606775e-004f, 5.3502177e-004f, 5.3397782e-004f, 5.3293592e-004f,
5.3189604e-004f, 5.3085820e-004f, 5.2982237e-004f, 5.2878858e-004f,
5.2775679e-004f, 5.2672702e-004f, 5.2569926e-004f, 5.2467351e-004f,
5.2364976e-004f, 5.2262800e-004f, 5.2160824e-004f, 5.2059047e-004f,
5.1957468e-004f, 5.1856088e-004f, 5.1754905e-004f, 5.1653920e-004f,
5.1553132e-004f, 5.1452541e-004f, 5.1352145e-004f, 5.1251946e-004f,
5.1151942e-004f, 5.1052134e-004f, 5.0952520e-004f, 5.0853100e-004f,
5.0753875e-004f, 5.0654843e-004f, 5.0556004e-004f, 5.0457358e-004f,
5.0358905e-004f, 5.0260644e-004f, 5.0162574e-004f, 5.0064696e-004f,
4.9967009e-004f, 4.9869512e-004f, 4.9772206e-004f, 4.9675089e-004f,
4.9578163e-004f, 4.9481425e-004f, 4.9384876e-004f, 4.9288515e-004f,
4.9192342e-004f, 4.9096357e-004f, 4.9000559e-004f, 4.8904949e-004f,
4.8809524e-004f, 4.8714286e-004f, 4.8619234e-004f, 4.8524367e-004f,
4.8429686e-004f, 4.8335189e-004f, 4.8240876e-004f, 4.8146748e-004f,
4.8052803e-004f, 4.7959041e-004f, 4.7865463e-004f, 4.7772067e-004f,
4.7678853e-004f, 4.7585821e-004f, 4.7492971e-004f, 4.7400302e-004f,
4.7307813e-004f, 4.7215505e-004f, 4.7123377e-004f, 4.7031429e-004f,
4.6939661e-004f, 4.6848071e-004f, 4.6756660e-004f, 4.6665428e-004f,
4.6574373e-004f, 4.6483497e-004f, 4.6392797e-004f, 4.6302275e-004f,
4.6211929e-004f, 4.6121759e-004f, 4.6031766e-004f, 4.5941948e-004f,
4.5852305e-004f, 4.5762837e-004f, 4.5673544e-004f, 4.5584424e-004f,
4.5495479e-004f, 4.5406708e-004f, 4.5318109e-004f, 4.5229684e-004f,
4.5141431e-004f, 4.5053350e-004f, 4.4965441e-004f, 4.4877703e-004f,
4.4790137e-004f, 4.4702742e-004f, 4.4615517e-004f, 4.4528462e-004f,
4.4441578e-004f, 4.4354862e-004f, 4.4268316e-004f, 4.4181939e-004f,
4.4095731e-004f, 4.4009690e-004f, 4.3923818e-004f, 4.3838113e-004f,
4.3752575e-004f, 4.3667204e-004f, 4.3582000e-004f, 4.3496962e-004f,
4.3412090e-004f, 4.3327383e-004f, 4.3242842e-004f, 4.3158466e-004f,
4.3074254e-004f, 4.2990207e-004f, 4.2906323e-004f, 4.2822604e-004f,
4.2739048e-004f, 4.2655654e-004f, 4.2572424e-004f, 4.2489356e-004f,
4.2406450e-004f, 4.2323705e-004f, 4.2241123e-004f, 4.2158701e-004f,
4.2076440e-004f, 4.1994340e-004f, 4.1912400e-004f, 4.1830619e-004f,
4.1748999e-004f, 4.1667537e-004f, 4.1586235e-004f, 4.1505091e-004f,
4.1424105e-004f, 4.1343278e-004f, 4.1262608e-004f, 4.1182096e-004f,
4.1101740e-004f, 4.1021542e-004f, 4.0941500e-004f, 4.0861614e-004f,
4.0781884e-004f, 4.0702310e-004f, 4.0622891e-004f, 4.0543626e-004f,
4.0464517e-004f, 4.0385562e-004f, 4.0306761e-004f, 4.0228113e-004f,
4.0149620e-004f, 4.0071279e-004f, 3.9993091e-004f, 3.9915056e-004f,
3.9837173e-004f, 3.9759442e-004f, 3.9681862e-004f, 3.9604434e-004f,
3.9527157e-004f, 3.9450031e-004f, 3.9373056e-004f, 3.9296230e-004f,
3.9219555e-004f, 3.9143029e-004f, 3.9066652e-004f, 3.8990424e-004f,
3.8914346e-004f, 3.8838415e-004f, 3.8762633e-004f, 3.8686999e-004f,

```
    3.8611512e-004f, 3.8536172e-004f, 3.8460980e-004f, 3.8385934e-004f,
    3.8311035e-004f, 3.8236281e-004f, 3.8161674e-004f, 3.8087212e-004f,
    3.8012896e-004f, 3.7938724e-004f, 3.7864698e-004f, 3.7790815e-004f,
    3.7717077e-004f, 3.7643483e-004f, 3.7570032e-004f, 3.7496725e-004f,
    3.7423560e-004f, 3.7350539e-004f, 3.7277660e-004f, 3.7204923e-004f,
    3.7132328e-004f, 3.7059875e-004f, 3.6987563e-004f, 3.6915392e-004f,
    3.6843362e-004f, 3.6771472e-004f, 3.6699723e-004f, 3.6628114e-004f,
    3.6556645e-004f, 3.6485315e-004f, 3.6414124e-004f, 3.6343072e-004f,
    3.6272159e-004f, 3.6201384e-004f, 3.6130747e-004f, 3.6060248e-004f,
    3.5989886e-004f, 3.5919662e-004f, 3.5849575e-004f, 3.5779625e-004f,
    3.5709811e-004f, 3.5640133e-004f, 3.5570591e-004f, 3.5501185e-004f,
    3.5431915e-004f, 3.5362779e-004f, 3.5293779e-004f, 3.5224913e-004f,
    3.5156182e-004f, 3.5087584e-004f, 3.5019121e-004f, 3.4950791e-004f,
    3.4882594e-004f, 3.4814530e-004f, 3.4746600e-004f, 3.4678801e-004f,
    3.4611135e-004f, 3.4543602e-004f, 3.4476199e-004f, 3.4408929e-004f,
    3.4341789e-004f, 3.4274781e-004f, 3.4207904e-004f, 3.4141156e-004f,
    3.4074540e-004f, 3.4008053e-004f, 3.3941696e-004f, 3.3875468e-004f,
    3.3809369e-004f, 3.3743400e-004f, 3.3677559e-004f, 3.3611847e-004f
};

static float exptable2 [EXPTABLE2] =
{
#if defined(_MSC_VER) && defined(_M_IX86)

    1.0000000e+000f, 9.9999905e-001f, 9.9999809e-001f, 9.9999714e-001f,
    9.9999619e-001f, 9.9999523e-001f, 9.9999428e-001f, 9.9999332e-001f,
    9.9999237e-001f, 9.9999142e-001f, 9.9999046e-001f, 9.9998951e-001f,
    9.9998856e-001f, 9.9998760e-001f, 9.9998665e-001f, 9.9998569e-001f,
    9.9998474e-001f, 9.9998379e-001f, 9.9998283e-001f, 9.9998188e-001f,
    9.9998093e-001f, 9.9997997e-001f, 9.9997902e-001f, 9.9997807e-001f,
    9.9997711e-001f, 9.9997616e-001f, 9.9997520e-001f, 9.9997425e-001f,
    9.9997330e-001f, 9.9997234e-001f, 9.9997139e-001f, 9.9997044e-001f,
    9.9996948e-001f, 9.9996853e-001f, 9.9996758e-001f, 9.9996662e-001f,
    9.9996567e-001f, 9.9996471e-001f, 9.9996376e-001f, 9.9996281e-001f,
    9.9996185e-001f, 9.9996090e-001f, 9.9995995e-001f, 9.9995899e-001f,
    9.9995804e-001f, 9.9995709e-001f, 9.9995613e-001f, 9.9995518e-001f,
    9.9995422e-001f, 9.9995327e-001f, 9.9995232e-001f, 9.9995136e-001f,
    9.9995041e-001f, 9.9994946e-001f, 9.9994850e-001f, 9.9994755e-001f,
    9.9994660e-001f, 9.9994564e-001f, 9.9994469e-001f, 9.9994373e-001f,
    9.9994278e-001f, 9.9994183e-001f, 9.9994087e-001f, 9.9993992e-001f,
    9.9993897e-001f, 9.9993801e-001f, 9.9993706e-001f, 9.9993611e-001f,
    9.9993515e-001f, 9.9993420e-001f, 9.9993325e-001f, 9.9993229e-001f,
    9.9993134e-001f, 9.9993038e-001f, 9.9992943e-001f, 9.9992848e-001f,
    9.9992752e-001f, 9.9992657e-001f, 9.9992562e-001f, 9.9992466e-001f,
    9.9992371e-001f, 9.9992276e-001f, 9.9992180e-001f, 9.9992085e-001f,
    9.9991989e-001f, 9.9991894e-001f, 9.9991799e-001f, 9.9991703e-001f,
    9.9991608e-001f, 9.9991513e-001f, 9.9991417e-001f, 9.9991322e-001f,
    9.9991227e-001f, 9.9991131e-001f, 9.9991036e-001f, 9.9990941e-001f,
    9.9990845e-001f, 9.9990750e-001f, 9.9990654e-001f, 9.9990559e-001f,
    9.9990464e-001f, 9.9990368e-001f, 9.9990273e-001f, 9.9990178e-001f,
    9.9990082e-001f, 9.9989987e-001f, 9.9989892e-001f, 9.9989796e-001f,
    9.9989701e-001f, 9.9989605e-001f, 9.9989510e-001f, 9.9989415e-001f,
    9.9989319e-001f, 9.9989224e-001f, 9.9989129e-001f, 9.9989033e-001f,
    9.9988938e-001f, 9.9988843e-001f, 9.9988747e-001f, 9.9988652e-001f,
    9.9988557e-001f, 9.9988461e-001f, 9.9988366e-001f, 9.9988270e-001f,
    9.9988175e-001f, 9.9988080e-001f, 9.9987984e-001f, 9.9987889e-001f,
    9.9987794e-001f, 9.9987698e-001f, 9.9987603e-001f, 9.9987508e-001f,
    9.9987412e-001f, 9.9987317e-001f, 9.9987222e-001f, 9.9987126e-001f,
    9.9987031e-001f, 9.9986936e-001f, 9.9986840e-001f, 9.9986745e-001f,
    9.9986649e-001f, 9.9986554e-001f, 9.9986459e-001f, 9.9986363e-001f,
    9.9986268e-001f, 9.9986173e-001f, 9.9986077e-001f, 9.9985982e-001f,
    9.9985887e-001f, 9.9985791e-001f, 9.9985696e-001f, 9.9985601e-001f,
    9.9985505e-001f, 9.9985410e-001f, 9.9985314e-001f, 9.9985219e-001f,
    9.9985124e-001f, 9.9985028e-001f, 9.9984933e-001f, 9.9984838e-001f,
    9.9984742e-001f, 9.9984647e-001f, 9.9984552e-001f, 9.9984456e-001f,
    9.9984361e-001f, 9.9984266e-001f, 9.9984170e-001f, 9.9984075e-001f,
    9.9983980e-001f, 9.9983884e-001f, 9.9983789e-001f, 9.9983693e-001f,
    9.9983598e-001f, 9.9983503e-001f, 9.9983407e-001f, 9.9983312e-001f,
    9.9983217e-001f, 9.9983121e-001f, 9.9983026e-001f, 9.9982931e-001f,
    9.9982835e-001f, 9.9982740e-001f, 9.9982645e-001f, 9.9982549e-001f,
    9.9982454e-001f, 9.9982359e-001f, 9.9982263e-001f, 9.9982168e-001f,
    9.9982073e-001f, 9.9981977e-001f, 9.9981882e-001f, 9.9981786e-001f,
    9.9981691e-001f, 9.9981596e-001f, 9.9981500e-001f, 9.9981405e-001f,
    9.9981310e-001f, 9.9981214e-001f, 9.9981119e-001f, 9.9981024e-001f,
    9.9980928e-001f, 9.9980833e-001f, 9.9980738e-001f, 9.9980642e-001f,
    9.9980547e-001f, 9.9980452e-001f, 9.9980356e-001f, 9.9980261e-001f,
```

9.9980166e-001f, 9.9980070e-001f, 9.9979975e-001f, 9.9979879e-001f,
9.9979784e-001f, 9.9979689e-001f, 9.9979593e-001f, 9.9979498e-001f,
9.9979403e-001f, 9.9979307e-001f, 9.9979212e-001f, 9.9979117e-001f,
9.9979021e-001f, 9.9978926e-001f, 9.9978831e-001f, 9.9978735e-001f,
9.9978640e-001f, 9.9978545e-001f, 9.9978449e-001f, 9.9978354e-001f,
9.9978259e-001f, 9.9978163e-001f, 9.9978068e-001f, 9.9977973e-001f,
9.9977877e-001f, 9.9977782e-001f, 9.9977687e-001f, 9.9977591e-001f,
9.9977496e-001f, 9.9977400e-001f, 9.9977305e-001f, 9.9977210e-001f,
9.9977114e-001f, 9.9977019e-001f, 9.9976924e-001f, 9.9976828e-001f,
9.9976733e-001f, 9.9976638e-001f, 9.9976542e-001f, 9.9976447e-001f,
9.9976352e-001f, 9.9976256e-001f, 9.9976161e-001f, 9.9976066e-001f,
9.9975970e-001f, 9.9975875e-001f, 9.9975780e-001f, 9.9975684e-001f,
9.9975589e-001f, 9.9975494e-001f, 9.9975398e-001f, 9.9975303e-001f,
9.9975208e-001f, 9.9975112e-001f, 9.9975017e-001f, 9.9974922e-001f,
9.9974826e-001f, 9.9974731e-001f, 9.9974635e-001f, 9.9974540e-001f,
9.9974445e-001f, 9.9974349e-001f, 9.9974254e-001f, 9.9974159e-001f,
9.9974063e-001f, 9.9973968e-001f, 9.9973873e-001f, 9.9973777e-001f,
9.9973682e-001f, 9.9973587e-001f, 9.9973491e-001f, 9.9973396e-001f,
9.9973301e-001f, 9.9973205e-001f, 9.9973110e-001f, 9.9973015e-001f,
9.9972919e-001f, 9.9972824e-001f, 9.9972729e-001f, 9.9972633e-001f,
9.9972538e-001f, 9.9972443e-001f, 9.9972347e-001f, 9.9972252e-001f,
9.9972157e-001f, 9.9972061e-001f, 9.9971966e-001f, 9.9971871e-001f,
9.9971775e-001f, 9.9971680e-001f, 9.9971585e-001f, 9.9971489e-001f,
9.9971394e-001f, 9.9971299e-001f, 9.9971203e-001f, 9.9971108e-001f,
9.9971013e-001f, 9.9970917e-001f, 9.9970822e-001f, 9.9970726e-001f,
9.9970631e-001f, 9.9970536e-001f, 9.9970440e-001f, 9.9970345e-001f,
9.9970250e-001f, 9.9970154e-001f, 9.9970059e-001f, 9.9969964e-001f,
9.9969868e-001f, 9.9969773e-001f, 9.9969678e-001f, 9.9969582e-001f,
9.9969487e-001f, 9.9969392e-001f, 9.9969296e-001f, 9.9969201e-001f,
9.9969106e-001f, 9.9969010e-001f, 9.9968915e-001f, 9.9968820e-001f,
9.9968724e-001f, 9.9968629e-001f, 9.9968534e-001f, 9.9968438e-001f,
9.9968343e-001f, 9.9968248e-001f, 9.9968152e-001f, 9.9968057e-001f,
9.9967962e-001f, 9.9967866e-001f, 9.9967771e-001f, 9.9967676e-001f,
9.9967580e-001f, 9.9967485e-001f, 9.9967390e-001f, 9.9967294e-001f,
9.9967199e-001f, 9.9967104e-001f, 9.9967008e-001f, 9.9966913e-001f,
9.9966818e-001f, 9.9966722e-001f, 9.9966627e-001f, 9.9966532e-001f,
9.9966436e-001f, 9.9966341e-001f, 9.9966246e-001f, 9.9966150e-001f,
9.9966055e-001f, 9.9965960e-001f, 9.9965864e-001f, 9.9965769e-001f,
9.9965674e-001f, 9.9965578e-001f, 9.9965483e-001f, 9.9965388e-001f,
9.9965292e-001f, 9.9965197e-001f, 9.9965102e-001f, 9.9965006e-001f,
9.9964911e-001f, 9.9964816e-001f, 9.9964720e-001f, 9.9964625e-001f,
9.9964530e-001f, 9.9964434e-001f, 9.9964339e-001f, 9.9964244e-001f,
9.9964148e-001f, 9.9964053e-001f, 9.9963958e-001f, 9.9963862e-001f,
9.9963767e-001f, 9.9963672e-001f, 9.9963576e-001f, 9.9963481e-001f,
9.9963386e-001f, 9.9963290e-001f, 9.9963195e-001f, 9.9963100e-001f,
9.9963004e-001f, 9.9962909e-001f, 9.9962814e-001f, 9.9962718e-001f,
9.9962623e-001f, 9.9962528e-001f, 9.9962432e-001f, 9.9962337e-001f,
9.9962242e-001f, 9.9962146e-001f, 9.9962051e-001f, 9.9961956e-001f,
9.9961860e-001f, 9.9961765e-001f, 9.9961670e-001f, 9.9961574e-001f,
9.9961479e-001f, 9.9961384e-001f, 9.9961288e-001f, 9.9961193e-001f,
9.9961098e-001f, 9.9961002e-001f, 9.9960907e-001f, 9.9960812e-001f,
9.9960716e-001f, 9.9960621e-001f, 9.9960526e-001f, 9.9960430e-001f,
9.9960335e-001f, 9.9960240e-001f, 9.9960144e-001f, 9.9960049e-001f,
9.9959954e-001f, 9.9959858e-001f, 9.9959763e-001f, 9.9959668e-001f,
9.9959572e-001f, 9.9959477e-001f, 9.9959382e-001f, 9.9959286e-001f,
9.9959191e-001f, 9.9959096e-001f, 9.9959000e-001f, 9.9958905e-001f,
9.9958810e-001f, 9.9958714e-001f, 9.9958619e-001f, 9.9958524e-001f,
9.9958428e-001f, 9.9958333e-001f, 9.9958238e-001f, 9.9958142e-001f,
9.9958047e-001f, 9.9957952e-001f, 9.9957856e-001f, 9.9957761e-001f,
9.9957666e-001f, 9.9957570e-001f, 9.9957475e-001f, 9.9957380e-001f,
9.9957285e-001f, 9.9957189e-001f, 9.9957094e-001f, 9.9956999e-001f,
9.9956903e-001f, 9.9956808e-001f, 9.9956713e-001f, 9.9956617e-001f,
9.9956522e-001f, 9.9956427e-001f, 9.9956331e-001f, 9.9956236e-001f,
9.9956141e-001f, 9.9956045e-001f, 9.9955950e-001f, 9.9955855e-001f,
9.9955759e-001f, 9.9955664e-001f, 9.9955569e-001f, 9.9955473e-001f,
9.9955378e-001f, 9.9955283e-001f, 9.9955187e-001f, 9.9955092e-001f,
9.9954997e-001f, 9.9954901e-001f, 9.9954806e-001f, 9.9954711e-001f,
9.9954615e-001f, 9.9954520e-001f, 9.9954425e-001f, 9.9954329e-001f,
9.9954234e-001f, 9.9954139e-001f, 9.9954043e-001f, 9.9953948e-001f,
9.9953853e-001f, 9.9953757e-001f, 9.9953662e-001f, 9.9953567e-001f,
9.9953472e-001f, 9.9953376e-001f, 9.9953281e-001f, 9.9953186e-001f,
9.9953090e-001f, 9.9952995e-001f, 9.9952900e-001f, 9.9952804e-001f,
9.9952709e-001f, 9.9952614e-001f, 9.9952518e-001f, 9.9952423e-001f,
9.9952328e-001f, 9.9952232e-001f, 9.9952137e-001f, 9.9952042e-001f,
9.9951946e-001f, 9.9951851e-001f, 9.9951756e-001f, 9.9951660e-001f,
9.9951565e-001f, 9.9951470e-001f, 9.9951374e-001f, 9.9951279e-001f,

```
9.9951184e-001f, 9.9951088e-001f, 9.9950993e-001f, 9.9950898e-001f,
9.9950803e-001f, 9.9950707e-001f, 9.9950612e-001f, 9.9950517e-001f,
9.9950421e-001f, 9.9950326e-001f, 9.9950231e-001f, 9.9950135e-001f,
9.9950040e-001f, 9.9949945e-001f, 9.9949849e-001f, 9.9949754e-001f,
9.9949659e-001f, 9.9949563e-001f, 9.9949468e-001f, 9.9949373e-001f,
9.9949277e-001f, 9.9949182e-001f, 9.9949087e-001f, 9.9948991e-001f,
9.9948896e-001f, 9.9948801e-001f, 9.9948705e-001f, 9.9948610e-001f,
9.9948515e-001f, 9.9948420e-001f, 9.9948324e-001f, 9.9948229e-001f,
9.9948134e-001f, 9.9948038e-001f, 9.9947943e-001f, 9.9947848e-001f,
9.9947752e-001f, 9.9947657e-001f, 9.9947562e-001f, 9.9947466e-001f,
9.9947371e-001f, 9.9947276e-001f, 9.9947180e-001f, 9.9947085e-001f,
9.9946990e-001f, 9.9946894e-001f, 9.9946799e-001f, 9.9946704e-001f,
9.9946608e-001f, 9.9946513e-001f, 9.9946418e-001f, 9.9946323e-001f,
9.9946227e-001f, 9.9946132e-001f, 9.9946037e-001f, 9.9945941e-001f,
9.9945846e-001f, 9.9945751e-001f, 9.9945655e-001f, 9.9945560e-001f,
9.9945465e-001f, 9.9945369e-001f, 9.9945274e-001f, 9.9945179e-001f,
9.9945083e-001f, 9.9944988e-001f, 9.9944893e-001f, 9.9944797e-001f,
9.9944702e-001f, 9.9944607e-001f, 9.9944512e-001f, 9.9944416e-001f,
9.9944321e-001f, 9.9944226e-001f, 9.9944130e-001f, 9.9944035e-001f,
9.9943940e-001f, 9.9943844e-001f, 9.9943749e-001f, 9.9943654e-001f,
9.9943558e-001f, 9.9943463e-001f, 9.9943368e-001f, 9.9943272e-001f,
9.9943177e-001f, 9.9943082e-001f, 9.9942987e-001f, 9.9942891e-001f,
9.9942796e-001f, 9.9942701e-001f, 9.9942605e-001f, 9.9942510e-001f,
9.9942415e-001f, 9.9942319e-001f, 9.9942224e-001f, 9.9942129e-001f,
9.9942033e-001f, 9.9941938e-001f, 9.9941843e-001f, 9.9941747e-001f,
9.9941652e-001f, 9.9941557e-001f, 9.9941462e-001f, 9.9941366e-001f,
9.9941271e-001f, 9.9941176e-001f, 9.9941080e-001f, 9.9940985e-001f,
9.9940890e-001f, 9.9940794e-001f, 9.9940699e-001f, 9.9940604e-001f,
9.9940508e-001f, 9.9940413e-001f, 9.9940318e-001f, 9.9940222e-001f,
9.9940127e-001f, 9.9940032e-001f, 9.9939937e-001f, 9.9939841e-001f,
9.9939746e-001f, 9.9939651e-001f, 9.9939555e-001f, 9.9939460e-001f,
9.9939365e-001f, 9.9939269e-001f, 9.9939174e-001f, 9.9939079e-001f,
9.9938983e-001f, 9.9938888e-001f, 9.9938793e-001f, 9.9938698e-001f,
9.9938602e-001f, 9.9938507e-001f, 9.9938412e-001f, 9.9938316e-001f,
9.9938221e-001f, 9.9938126e-001f, 9.9938030e-001f, 9.9937935e-001f,
9.9937840e-001f, 9.9937744e-001f, 9.9937649e-001f, 9.9937554e-001f,
9.9937459e-001f, 9.9937363e-001f, 9.9937268e-001f, 9.9937173e-001f,
9.9937077e-001f, 9.9936982e-001f, 9.9936887e-001f, 9.9936791e-001f,
9.9936696e-001f, 9.9936601e-001f, 9.9936505e-001f, 9.9936410e-001f,
9.9936315e-001f, 9.9936220e-001f, 9.9936124e-001f, 9.9936029e-001f,
9.9935934e-001f, 9.9935838e-001f, 9.9935743e-001f, 9.9935648e-001f,
9.9935552e-001f, 9.9935457e-001f, 9.9935362e-001f, 9.9935266e-001f,
9.9935171e-001f, 9.9935076e-001f, 9.9934981e-001f, 9.9934885e-001f,
9.9934790e-001f, 9.9934695e-001f, 9.9934599e-001f, 9.9934504e-001f,
9.9934409e-001f, 9.9934313e-001f, 9.9934218e-001f, 9.9934123e-001f,
9.9934028e-001f, 9.9933932e-001f, 9.9933837e-001f, 9.9933742e-001f,
9.9933646e-001f, 9.9933551e-001f, 9.9933456e-001f, 9.9933360e-001f,
9.9933265e-001f, 9.9933170e-001f, 9.9933074e-001f, 9.9932979e-001f,
9.9932884e-001f, 9.9932789e-001f, 9.9932693e-001f, 9.9932598e-001f,
9.9932503e-001f, 9.9932407e-001f, 9.9932312e-001f, 9.9932217e-001f,
9.9932121e-001f, 9.9932026e-001f, 9.9931931e-001f, 9.9931836e-001f,
9.9931740e-001f, 9.9931645e-001f, 9.9931550e-001f, 9.9931454e-001f,
9.9931359e-001f, 9.9931264e-001f, 9.9931168e-001f, 9.9931073e-001f,
9.9930978e-001f, 9.9930883e-001f, 9.9930787e-001f, 9.9930692e-001f,
9.9930597e-001f, 9.9930501e-001f, 9.9930406e-001f, 9.9930311e-001f,
9.9930215e-001f, 9.9930120e-001f, 9.9930025e-001f, 9.9929929e-001f,
9.9929834e-001f, 9.9929739e-001f, 9.9929644e-001f, 9.9929548e-001f,
9.9929453e-001f, 9.9929358e-001f, 9.9929262e-001f, 9.9929167e-001f,
9.9929072e-001f, 9.9928976e-001f, 9.9928881e-001f, 9.9928786e-001f,
9.9928691e-001f, 9.9928595e-001f, 9.9928500e-001f, 9.9928405e-001f,
9.9928309e-001f, 9.9928214e-001f, 9.9928119e-001f, 9.9928024e-001f,
9.9927928e-001f, 9.9927833e-001f, 9.9927738e-001f, 9.9927642e-001f,
9.9927547e-001f, 9.9927452e-001f, 9.9927356e-001f, 9.9927261e-001f,
9.9927166e-001f, 9.9927071e-001f, 9.9926975e-001f, 9.9926880e-001f,
9.9926785e-001f, 9.9926689e-001f, 9.9926594e-001f, 9.9926499e-001f,
9.9926403e-001f, 9.9926308e-001f, 9.9926213e-001f, 9.9926118e-001f,
9.9926022e-001f, 9.9925927e-001f, 9.9925832e-001f, 9.9925736e-001f,
9.9925641e-001f, 9.9925546e-001f, 9.9925450e-001f, 9.9925355e-001f,
9.9925260e-001f, 9.9925165e-001f, 9.9925069e-001f, 9.9924974e-001f,
9.9924879e-001f, 9.9924783e-001f, 9.9924688e-001f, 9.9924593e-001f,
9.9924498e-001f, 9.9924402e-001f, 9.9924307e-001f, 9.9924212e-001f,
9.9924116e-001f, 9.9924021e-001f, 9.9923926e-001f, 9.9923830e-001f,
9.9923735e-001f, 9.9923640e-001f, 9.9923545e-001f, 9.9923449e-001f,
9.9923354e-001f, 9.9923259e-001f, 9.9923163e-001f, 9.9923068e-001f,
9.9922973e-001f, 9.9922878e-001f, 9.9922782e-001f, 9.9922687e-001f,
9.9922592e-001f, 9.9922496e-001f, 9.9922401e-001f, 9.9922306e-001f,
```

9.9922210e-001f, 9.9922115e-001f, 9.9922020e-001f, 9.9921925e-001f,
9.9921829e-001f, 9.9921734e-001f, 9.9921639e-001f, 9.9921543e-001f,
9.9921448e-001f, 9.9921353e-001f, 9.9921258e-001f, 9.9921162e-001f,
9.9921067e-001f, 9.9920972e-001f, 9.9920876e-001f, 9.9920781e-001f,
9.9920686e-001f, 9.9920590e-001f, 9.9920495e-001f, 9.9920400e-001f,
9.9920305e-001f, 9.9920209e-001f, 9.9920114e-001f, 9.9920019e-001f,
9.9919923e-001f, 9.9919828e-001f, 9.9919733e-001f, 9.9919638e-001f,
9.9919542e-001f, 9.9919447e-001f, 9.9919352e-001f, 9.9919256e-001f,
9.9919161e-001f, 9.9919066e-001f, 9.9918971e-001f, 9.9918875e-001f,
9.9918780e-001f, 9.9918685e-001f, 9.9918589e-001f, 9.9918494e-001f,
9.9918399e-001f, 9.9918304e-001f, 9.9918208e-001f, 9.9918113e-001f,
9.9918018e-001f, 9.9917922e-001f, 9.9917827e-001f, 9.9917732e-001f,
9.9917636e-001f, 9.9917541e-001f, 9.9917446e-001f, 9.9917351e-001f,
9.9917255e-001f, 9.9917160e-001f, 9.9917065e-001f, 9.9916969e-001f,
9.9916874e-001f, 9.9916779e-001f, 9.9916684e-001f, 9.9916588e-001f,
9.9916493e-001f, 9.9916398e-001f, 9.9916302e-001f, 9.9916207e-001f,
9.9916112e-001f, 9.9916017e-001f, 9.9915921e-001f, 9.9915826e-001f,
9.9915731e-001f, 9.9915635e-001f, 9.9915540e-001f, 9.9915445e-001f,
9.9915350e-001f, 9.9915254e-001f, 9.9915159e-001f, 9.9915064e-001f,
9.9914968e-001f, 9.9914873e-001f, 9.9914778e-001f, 9.9914683e-001f,
9.9914587e-001f, 9.9914492e-001f, 9.9914397e-001f, 9.9914301e-001f,
9.9914206e-001f, 9.9914111e-001f, 9.9914016e-001f, 9.9913920e-001f,
9.9913825e-001f, 9.9913730e-001f, 9.9913634e-001f, 9.9913539e-001f,
9.9913444e-001f, 9.9913349e-001f, 9.9913253e-001f, 9.9913158e-001f,
9.9913063e-001f, 9.9912967e-001f, 9.9912872e-001f, 9.9912777e-001f,
9.9912682e-001f, 9.9912586e-001f, 9.9912491e-001f, 9.9912396e-001f,
9.9912300e-001f, 9.9912205e-001f, 9.9912110e-001f, 9.9912015e-001f,
9.9911919e-001f, 9.9911824e-001f, 9.9911729e-001f, 9.9911633e-001f,
9.9911538e-001f, 9.9911443e-001f, 9.9911348e-001f, 9.9911252e-001f,
9.9911157e-001f, 9.9911062e-001f, 9.9910966e-001f, 9.9910871e-001f,
9.9910776e-001f, 9.9910681e-001f, 9.9910585e-001f, 9.9910490e-001f,
9.9910395e-001f, 9.9910300e-001f, 9.9910204e-001f, 9.9910109e-001f,
9.9910014e-001f, 9.9909918e-001f, 9.9909823e-001f, 9.9909728e-001f,
9.9909633e-001f, 9.9909537e-001f, 9.9909442e-001f, 9.9909347e-001f,
9.9909251e-001f, 9.9909156e-001f, 9.9909061e-001f, 9.9908966e-001f,
9.9908870e-001f, 9.9908775e-001f, 9.9908680e-001f, 9.9908584e-001f,
9.9908489e-001f, 9.9908394e-001f, 9.9908299e-001f, 9.9908203e-001f,
9.9908108e-001f, 9.9908013e-001f, 9.9907917e-001f, 9.9907822e-001f,
9.9907727e-001f, 9.9907632e-001f, 9.9907536e-001f, 9.9907441e-001f,
9.9907346e-001f, 9.9907251e-001f, 9.9907155e-001f, 9.9907060e-001f,
9.9906965e-001f, 9.9906869e-001f, 9.9906774e-001f, 9.9906679e-001f,
9.9906584e-001f, 9.9906488e-001f, 9.9906393e-001f, 9.9906298e-001f,
9.9906202e-001f, 9.9906107e-001f, 9.9906012e-001f, 9.9905917e-001f,
9.9905821e-001f, 9.9905726e-001f, 9.9905631e-001f, 9.9905536e-001f,
9.9905440e-001f, 9.9905345e-001f, 9.9905250e-001f, 9.9905154e-001f,
9.9905059e-001f, 9.9904964e-001f, 9.9904869e-001f, 9.9904773e-001f,
9.9904678e-001f, 9.9904583e-001f, 9.9904487e-001f, 9.9904392e-001f,
9.9904297e-001f, 9.9904202e-001f, 9.9904106e-001f, 9.9904011e-001f,
9.9903916e-001f, 9.9903821e-001f, 9.9903725e-001f, 9.9903630e-001f,
9.9903535e-001f, 9.9903439e-001f, 9.9903344e-001f, 9.9903249e-001f,
9.9903154e-001f, 9.9903058e-001f, 9.9902963e-001f, 9.9902868e-001f,
9.9902773e-001f, 9.9902677e-001f, 9.9902582e-001f, 9.9902487e-001f,
9.9902391e-001f, 9.9902296e-001f, 9.9902201e-001f, 9.9902106e-001f,
9.9902010e-001f, 9.9901915e-001f, 9.9901820e-001f, 9.9901725e-001f,
9.9901629e-001f, 9.9901534e-001f, 9.9901439e-001f, 9.9901343e-001f,
9.9901248e-001f, 9.9901153e-001f, 9.9901058e-001f, 9.9900962e-001f,
9.9900867e-001f, 9.9900772e-001f, 9.9900676e-001f, 9.9900581e-001f,
9.9900486e-001f, 9.9900391e-001f, 9.9900295e-001f, 9.9900200e-001f,
9.9900105e-001f, 9.9900010e-001f, 9.9899914e-001f, 9.9899819e-001f,
9.9899724e-001f, 9.9899629e-001f, 9.9899533e-001f, 9.9899438e-001f,
9.9899343e-001f, 9.9899247e-001f, 9.9899152e-001f, 9.9899057e-001f,
9.9898962e-001f, 9.9898866e-001f, 9.9898771e-001f, 9.9898676e-001f,
9.9898581e-001f, 9.9898485e-001f, 9.9898390e-001f, 9.9898295e-001f,
9.9898199e-001f, 9.9898104e-001f, 9.9898009e-001f, 9.9897914e-001f,
9.9897818e-001f, 9.9897723e-001f, 9.9897628e-001f, 9.9897533e-001f,
9.9897437e-001f, 9.9897342e-001f, 9.9897247e-001f, 9.9897151e-001f,
9.9897056e-001f, 9.9896961e-001f, 9.9896866e-001f, 9.9896770e-001f,
9.9896675e-001f, 9.9896580e-001f, 9.9896485e-001f, 9.9896389e-001f,
9.9896294e-001f, 9.9896199e-001f, 9.9896104e-001f, 9.9896008e-001f,
9.9895913e-001f, 9.9895818e-001f, 9.9895722e-001f, 9.9895627e-001f,
9.9895532e-001f, 9.9895437e-001f, 9.9895341e-001f, 9.9895246e-001f,
9.9895151e-001f, 9.9895056e-001f, 9.9894960e-001f, 9.9894865e-001f,
9.9894770e-001f, 9.9894674e-001f, 9.9894579e-001f, 9.9894484e-001f,
9.9894389e-001f, 9.9894293e-001f, 9.9894198e-001f, 9.9894103e-001f,
9.9894008e-001f, 9.9893912e-001f, 9.9893817e-001f, 9.9893722e-001f,
9.9893627e-001f, 9.9893531e-001f, 9.9893436e-001f, 9.9893341e-001f,

9.9893245e-001f, 9.9893150e-001f, 9.9893055e-001f, 9.9892960e-001f,
9.9892864e-001f, 9.9892769e-001f, 9.9892674e-001f, 9.9892579e-001f,
9.9892483e-001f, 9.9892388e-001f, 9.9892293e-001f, 9.9892198e-001f,
9.9892102e-001f, 9.9892007e-001f, 9.9891912e-001f, 9.9891817e-001f,
9.9891721e-001f, 9.9891626e-001f, 9.9891531e-001f, 9.9891435e-001f,
9.9891340e-001f, 9.9891245e-001f, 9.9891150e-001f, 9.9891054e-001f,
9.9890959e-001f, 9.9890864e-001f, 9.9890769e-001f, 9.9890673e-001f,
9.9890578e-001f, 9.9890483e-001f, 9.9890388e-001f, 9.9890292e-001f,
9.9890197e-001f, 9.9890102e-001f, 9.9890007e-001f, 9.9889911e-001f,
9.9889816e-001f, 9.9889721e-001f, 9.9889625e-001f, 9.9889530e-001f,
9.9889435e-001f, 9.9889340e-001f, 9.9889244e-001f, 9.9889149e-001f,
9.9889054e-001f, 9.9888959e-001f, 9.9888863e-001f, 9.9888768e-001f,
9.9888673e-001f, 9.9888578e-001f, 9.9888482e-001f, 9.9888387e-001f,
9.9888292e-001f, 9.9888197e-001f, 9.9888101e-001f, 9.9888006e-001f,
9.9887911e-001f, 9.9887816e-001f, 9.9887720e-001f, 9.9887625e-001f,
9.9887530e-001f, 9.9887434e-001f, 9.9887339e-001f, 9.9887244e-001f,
9.9887149e-001f, 9.9887053e-001f, 9.9886958e-001f, 9.9886863e-001f,
9.9886768e-001f, 9.9886672e-001f, 9.9886577e-001f, 9.9886482e-001f,
9.9886387e-001f, 9.9886291e-001f, 9.9886196e-001f, 9.9886101e-001f,
9.9886006e-001f, 9.9885910e-001f, 9.9885815e-001f, 9.9885720e-001f,
9.9885625e-001f, 9.9885529e-001f, 9.9885434e-001f, 9.9885339e-001f,
9.9885244e-001f, 9.9885148e-001f, 9.9885053e-001f, 9.9884958e-001f,
9.9884862e-001f, 9.9884767e-001f, 9.9884672e-001f, 9.9884577e-001f,
9.9884481e-001f, 9.9884386e-001f, 9.9884291e-001f, 9.9884196e-001f,
9.9884100e-001f, 9.9884005e-001f, 9.9883910e-001f, 9.9883815e-001f,
9.9883719e-001f, 9.9883624e-001f, 9.9883529e-001f, 9.9883434e-001f,
9.9883338e-001f, 9.9883243e-001f, 9.9883148e-001f, 9.9883053e-001f,
9.9882957e-001f, 9.9882862e-001f, 9.9882767e-001f, 9.9882672e-001f,
9.9882576e-001f, 9.9882481e-001f, 9.9882386e-001f, 9.9882291e-001f,
9.9882195e-001f, 9.9882100e-001f, 9.9882005e-001f, 9.9881910e-001f,
9.9881814e-001f, 9.9881719e-001f, 9.9881624e-001f, 9.9881529e-001f,
9.9881433e-001f, 9.9881338e-001f, 9.9881243e-001f, 9.9881147e-001f,
9.9881052e-001f, 9.9880957e-001f, 9.9880862e-001f, 9.9880766e-001f,
9.9880671e-001f, 9.9880576e-001f, 9.9880481e-001f, 9.9880385e-001f,
9.9880290e-001f, 9.9880195e-001f, 9.9880100e-001f, 9.9880004e-001f,
9.9879909e-001f, 9.9879814e-001f, 9.9879719e-001f, 9.9879623e-001f,
9.9879528e-001f, 9.9879433e-001f, 9.9879338e-001f, 9.9879242e-001f,
9.9879147e-001f, 9.9879052e-001f, 9.9878957e-001f, 9.9878861e-001f,
9.9878766e-001f, 9.9878671e-001f, 9.9878576e-001f, 9.9878480e-001f,
9.9878385e-001f, 9.9878290e-001f, 9.9878195e-001f, 9.9878099e-001f,
9.9878004e-001f, 9.9877909e-001f, 9.9877814e-001f, 9.9877718e-001f,
9.9877623e-001f, 9.9877528e-001f, 9.9877433e-001f, 9.9877337e-001f,
9.9877242e-001f, 9.9877147e-001f, 9.9877052e-001f, 9.9876956e-001f,
9.9876861e-001f, 9.9876766e-001f, 9.9876671e-001f, 9.9876575e-001f,
9.9876480e-001f, 9.9876385e-001f, 9.9876290e-001f, 9.9876194e-001f,
9.9876099e-001f, 9.9876004e-001f, 9.9875909e-001f, 9.9875813e-001f,
9.9875718e-001f, 9.9875623e-001f, 9.9875528e-001f, 9.9875432e-001f,
9.9875337e-001f, 9.9875242e-001f, 9.9875147e-001f, 9.9875051e-001f,
9.9874956e-001f, 9.9874861e-001f, 9.9874766e-001f, 9.9874670e-001f,
9.9874575e-001f, 9.9874480e-001f, 9.9874385e-001f, 9.9874289e-001f,
9.9874194e-001f, 9.9874099e-001f, 9.9874004e-001f, 9.9873908e-001f,
9.9873813e-001f, 9.9873718e-001f, 9.9873623e-001f, 9.9873527e-001f,
9.9873432e-001f, 9.9873337e-001f, 9.9873242e-001f, 9.9873146e-001f,
9.9873051e-001f, 9.9872956e-001f, 9.9872861e-001f, 9.9872765e-001f,
9.9872670e-001f, 9.9872575e-001f, 9.9872480e-001f, 9.9872385e-001f,
9.9872289e-001f, 9.9872194e-001f, 9.9872099e-001f, 9.9872004e-001f,
9.9871908e-001f, 9.9871813e-001f, 9.9871718e-001f, 9.9871623e-001f,
9.9871527e-001f, 9.9871432e-001f, 9.9871337e-001f, 9.9871242e-001f,
9.9871146e-001f, 9.9871051e-001f, 9.9870956e-001f, 9.9870861e-001f,
9.9870765e-001f, 9.9870670e-001f, 9.9870575e-001f, 9.9870480e-001f,
9.9870384e-001f, 9.9870289e-001f, 9.9870194e-001f, 9.9870099e-001f,
9.9870003e-001f, 9.9869908e-001f, 9.9869813e-001f, 9.9869718e-001f,
9.9869622e-001f, 9.9869527e-001f, 9.9869432e-001f, 9.9869337e-001f,
9.9869241e-001f, 9.9869146e-001f, 9.9869051e-001f, 9.9868956e-001f,
9.9868860e-001f, 9.9868765e-001f, 9.9868670e-001f, 9.9868575e-001f,
9.9868480e-001f, 9.9868384e-001f, 9.9868289e-001f, 9.9868194e-001f,
9.9868099e-001f, 9.9868003e-001f, 9.9867908e-001f, 9.9867813e-001f,
9.9867718e-001f, 9.9867622e-001f, 9.9867527e-001f, 9.9867432e-001f,
9.9867337e-001f, 9.9867241e-001f, 9.9867146e-001f, 9.9867051e-001f,
9.9866956e-001f, 9.9866860e-001f, 9.9866765e-001f, 9.9866670e-001f,
9.9866575e-001f, 9.9866479e-001f, 9.9866384e-001f, 9.9866289e-001f,
9.9866194e-001f, 9.9866098e-001f, 9.9866003e-001f, 9.9865908e-001f,
9.9865813e-001f, 9.9865718e-001f, 9.9865622e-001f, 9.9865527e-001f,
9.9865432e-001f, 9.9865337e-001f, 9.9865241e-001f, 9.9865146e-001f,
9.9865051e-001f, 9.9864956e-001f, 9.9864860e-001f, 9.9864765e-001f,
9.9864670e-001f, 9.9864575e-001f, 9.9864479e-001f, 9.9864384e-001f,

9.9864289e-001f, 9.9864194e-001f, 9.9864098e-001f, 9.9864003e-001f,
9.9863908e-001f, 9.9863813e-001f, 9.9863718e-001f, 9.9863622e-001f,
9.9863527e-001f, 9.9863432e-001f, 9.9863337e-001f, 9.9863241e-001f,
9.9863146e-001f, 9.9863051e-001f, 9.9862956e-001f, 9.9862860e-001f,
9.9862765e-001f, 9.9862670e-001f, 9.9862575e-001f, 9.9862479e-001f,
9.9862384e-001f, 9.9862289e-001f, 9.9862194e-001f, 9.9862098e-001f,
9.9862003e-001f, 9.9861908e-001f, 9.9861813e-001f, 9.9861718e-001f,
9.9861622e-001f, 9.9861527e-001f, 9.9861432e-001f, 9.9861337e-001f,
9.9861241e-001f, 9.9861146e-001f, 9.9861051e-001f, 9.9860956e-001f,
9.9860860e-001f, 9.9860765e-001f, 9.9860670e-001f, 9.9860575e-001f,
9.9860480e-001f, 9.9860384e-001f, 9.9860289e-001f, 9.9860194e-001f,
9.9860099e-001f, 9.9860003e-001f, 9.9859908e-001f, 9.9859813e-001f,
9.9859718e-001f, 9.9859622e-001f, 9.9859527e-001f, 9.9859432e-001f,
9.9859337e-001f, 9.9859241e-001f, 9.9859146e-001f, 9.9859051e-001f,
9.9858956e-001f, 9.9858861e-001f, 9.9858765e-001f, 9.9858670e-001f,
9.9858575e-001f, 9.9858480e-001f, 9.9858384e-001f, 9.9858289e-001f,
9.9858194e-001f, 9.9858099e-001f, 9.9858003e-001f, 9.9857908e-001f,
9.9857813e-001f, 9.9857718e-001f, 9.9857623e-001f, 9.9857527e-001f,
9.9857432e-001f, 9.9857337e-001f, 9.9857242e-001f, 9.9857146e-001f,
9.9857051e-001f, 9.9856956e-001f, 9.9856861e-001f, 9.9856765e-001f,
9.9856670e-001f, 9.9856575e-001f, 9.9856480e-001f, 9.9856385e-001f,
9.9856289e-001f, 9.9856194e-001f, 9.9856099e-001f, 9.9856004e-001f,
9.9855908e-001f, 9.9855813e-001f, 9.9855718e-001f, 9.9855623e-001f,
9.9855527e-001f, 9.9855432e-001f, 9.9855337e-001f, 9.9855242e-001f,
9.9855147e-001f, 9.9855051e-001f, 9.9854956e-001f, 9.9854861e-001f,
9.9854766e-001f, 9.9854670e-001f, 9.9854575e-001f, 9.9854480e-001f,
9.9854385e-001f, 9.9854289e-001f, 9.9854194e-001f, 9.9854099e-001f,
9.9854004e-001f, 9.9853909e-001f, 9.9853813e-001f, 9.9853718e-001f,
9.9853623e-001f, 9.9853528e-001f, 9.9853432e-001f, 9.9853337e-001f,
9.9853242e-001f, 9.9853147e-001f, 9.9853051e-001f, 9.9852956e-001f,
9.9852861e-001f, 9.9852766e-001f, 9.9852671e-001f, 9.9852575e-001f,
9.9852480e-001f, 9.9852385e-001f, 9.9852290e-001f, 9.9852194e-001f,
9.9852099e-001f, 9.9852004e-001f, 9.9851909e-001f, 9.9851814e-001f,
9.9851718e-001f, 9.9851623e-001f, 9.9851528e-001f, 9.9851433e-001f,
9.9851337e-001f, 9.9851242e-001f, 9.9851147e-001f, 9.9851052e-001f,
9.9850957e-001f, 9.9850861e-001f, 9.9850766e-001f, 9.9850671e-001f,
9.9850576e-001f, 9.9850480e-001f, 9.9850385e-001f, 9.9850290e-001f,
9.9850195e-001f, 9.9850099e-001f, 9.9850004e-001f, 9.9849909e-001f,
9.9849814e-001f, 9.9849719e-001f, 9.9849623e-001f, 9.9849528e-001f,
9.9849433e-001f, 9.9849338e-001f, 9.9849242e-001f, 9.9849147e-001f,
9.9849052e-001f, 9.9848957e-001f, 9.9848862e-001f, 9.9848766e-001f,
9.9848671e-001f, 9.9848576e-001f, 9.9848481e-001f, 9.9848385e-001f,
9.9848290e-001f, 9.9848195e-001f, 9.9848100e-001f, 9.9848005e-001f,
9.9847909e-001f, 9.9847814e-001f, 9.9847719e-001f, 9.9847624e-001f,
9.9847528e-001f, 9.9847433e-001f, 9.9847338e-001f, 9.9847243e-001f,
9.9847148e-001f, 9.9847052e-001f, 9.9846957e-001f, 9.9846862e-001f,
9.9846767e-001f, 9.9846671e-001f, 9.9846576e-001f, 9.9846481e-001f,
9.9846386e-001f, 9.9846291e-001f, 9.9846195e-001f, 9.9846100e-001f,
9.9846005e-001f, 9.9845910e-001f, 9.9845814e-001f, 9.9845719e-001f,
9.9845624e-001f, 9.9845529e-001f, 9.9845434e-001f, 9.9845338e-001f,
9.9845243e-001f, 9.9845148e-001f, 9.9845053e-001f, 9.9844958e-001f,
9.9844862e-001f, 9.9844767e-001f, 9.9844672e-001f, 9.9844577e-001f,
9.9844481e-001f, 9.9844386e-001f, 9.9844291e-001f, 9.9844196e-001f,
9.9844101e-001f, 9.9844005e-001f, 9.9843910e-001f, 9.9843815e-001f,
9.9843720e-001f, 9.9843624e-001f, 9.9843529e-001f, 9.9843434e-001f,
9.9843339e-001f, 9.9843244e-001f, 9.9843148e-001f, 9.9843053e-001f,
9.9842958e-001f, 9.9842863e-001f, 9.9842767e-001f, 9.9842672e-001f,
9.9842577e-001f, 9.9842482e-001f, 9.9842387e-001f, 9.9842291e-001f,
9.9842196e-001f, 9.9842101e-001f, 9.9842006e-001f, 9.9841911e-001f,
9.9841815e-001f, 9.9841720e-001f, 9.9841625e-001f, 9.9841530e-001f,
9.9841434e-001f, 9.9841339e-001f, 9.9841244e-001f, 9.9841149e-001f,
9.9841054e-001f, 9.9840958e-001f, 9.9840863e-001f, 9.9840768e-001f,
9.9840673e-001f, 9.9840577e-001f, 9.9840482e-001f, 9.9840387e-001f,
9.9840292e-001f, 9.9840197e-001f, 9.9840101e-001f, 9.9840006e-001f,
9.9839911e-001f, 9.9839816e-001f, 9.9839721e-001f, 9.9839625e-001f,
9.9839530e-001f, 9.9839435e-001f, 9.9839340e-001f, 9.9839244e-001f,
9.9839149e-001f, 9.9839054e-001f, 9.9838959e-001f, 9.9838864e-001f,
9.9838768e-001f, 9.9838673e-001f, 9.9838578e-001f, 9.9838483e-001f,
9.9838388e-001f, 9.9838292e-001f, 9.9838197e-001f, 9.9838102e-001f,
9.9838007e-001f, 9.9837912e-001f, 9.9837816e-001f, 9.9837721e-001f,
9.9837626e-001f, 9.9837531e-001f, 9.9837435e-001f, 9.9837340e-001f,
9.9837245e-001f, 9.9837150e-001f, 9.9837055e-001f, 9.9836959e-001f,
9.9836864e-001f, 9.9836769e-001f, 9.9836674e-001f, 9.9836579e-001f,
9.9836483e-001f, 9.9836388e-001f, 9.9836293e-001f, 9.9836198e-001f,
9.9836102e-001f, 9.9836007e-001f, 9.9835912e-001f, 9.9835817e-001f,
9.9835722e-001f, 9.9835626e-001f, 9.9835531e-001f, 9.9835436e-001f,

9.9835341e-001f, 9.9835246e-001f, 9.9835150e-001f, 9.9835055e-001f,
9.9834960e-001f, 9.9834865e-001f, 9.9834770e-001f, 9.9834674e-001f,
9.9834579e-001f, 9.9834484e-001f, 9.9834389e-001f, 9.9834293e-001f,
9.9834198e-001f, 9.9834103e-001f, 9.9834008e-001f, 9.9833913e-001f,
9.9833817e-001f, 9.9833722e-001f, 9.9833627e-001f, 9.9833532e-001f,
9.9833437e-001f, 9.9833341e-001f, 9.9833246e-001f, 9.9833151e-001f,
9.9833056e-001f, 9.9832961e-001f, 9.9832865e-001f, 9.9832770e-001f,
9.9832675e-001f, 9.9832580e-001f, 9.9832485e-001f, 9.9832389e-001f,
9.9832294e-001f, 9.9832199e-001f, 9.9832104e-001f, 9.9832008e-001f,
9.9831913e-001f, 9.9831818e-001f, 9.9831723e-001f, 9.9831628e-001f,
9.9831532e-001f, 9.9831437e-001f, 9.9831342e-001f, 9.9831247e-001f,
9.9831152e-001f, 9.9831056e-001f, 9.9830961e-001f, 9.9830866e-001f,
9.9830771e-001f, 9.9830676e-001f, 9.9830580e-001f, 9.9830485e-001f,
9.9830390e-001f, 9.9830295e-001f, 9.9830200e-001f, 9.9830104e-001f,
9.9830009e-001f, 9.9829914e-001f, 9.9829819e-001f, 9.9829724e-001f,
9.9829628e-001f, 9.9829533e-001f, 9.9829438e-001f, 9.9829343e-001f,
9.9829248e-001f, 9.9829152e-001f, 9.9829057e-001f, 9.9828962e-001f,
9.9828867e-001f, 9.9828771e-001f, 9.9828676e-001f, 9.9828581e-001f,
9.9828486e-001f, 9.9828391e-001f, 9.9828295e-001f, 9.9828200e-001f,
9.9828105e-001f, 9.9828010e-001f, 9.9827915e-001f, 9.9827819e-001f,
9.9827724e-001f, 9.9827629e-001f, 9.9827534e-001f, 9.9827439e-001f,
9.9827343e-001f, 9.9827248e-001f, 9.9827153e-001f, 9.9827058e-001f,
9.9826963e-001f, 9.9826867e-001f, 9.9826772e-001f, 9.9826677e-001f,
9.9826582e-001f, 9.9826487e-001f, 9.9826391e-001f, 9.9826296e-001f,
9.9826201e-001f, 9.9826106e-001f, 9.9826011e-001f, 9.9825915e-001f,
9.9825820e-001f, 9.9825725e-001f, 9.9825630e-001f, 9.9825535e-001f,
9.9825439e-001f, 9.9825344e-001f, 9.9825249e-001f, 9.9825154e-001f,
9.9825059e-001f, 9.9824963e-001f, 9.9824868e-001f, 9.9824773e-001f,
9.9824678e-001f, 9.9824583e-001f, 9.9824487e-001f, 9.9824392e-001f,
9.9824297e-001f, 9.9824202e-001f, 9.9824107e-001f, 9.9824011e-001f,
9.9823916e-001f, 9.9823821e-001f, 9.9823726e-001f, 9.9823631e-001f,
9.9823535e-001f, 9.9823440e-001f, 9.9823345e-001f, 9.9823250e-001f,
9.9823155e-001f, 9.9823059e-001f, 9.9822964e-001f, 9.9822869e-001f,
9.9822774e-001f, 9.9822679e-001f, 9.9822583e-001f, 9.9822488e-001f,
9.9822393e-001f, 9.9822298e-001f, 9.9822203e-001f, 9.9822107e-001f,
9.9822012e-001f, 9.9821917e-001f, 9.9821822e-001f, 9.9821727e-001f,
9.9821631e-001f, 9.9821536e-001f, 9.9821441e-001f, 9.9821346e-001f,
9.9821251e-001f, 9.9821155e-001f, 9.9821060e-001f, 9.9820965e-001f,
9.9820870e-001f, 9.9820775e-001f, 9.9820679e-001f, 9.9820584e-001f,
9.9820489e-001f, 9.9820394e-001f, 9.9820299e-001f, 9.9820203e-001f,
9.9820108e-001f, 9.9820013e-001f, 9.9819918e-001f, 9.9819823e-001f,
9.9819728e-001f, 9.9819632e-001f, 9.9819537e-001f, 9.9819442e-001f,
9.9819347e-001f, 9.9819252e-001f, 9.9819156e-001f, 9.9819061e-001f,
9.9818966e-001f, 9.9818871e-001f, 9.9818776e-001f, 9.9818680e-001f,
9.9818585e-001f, 9.9818490e-001f, 9.9818395e-001f, 9.9818300e-001f,
9.9818204e-001f, 9.9818109e-001f, 9.9818014e-001f, 9.9817919e-001f,
9.9817824e-001f, 9.9817728e-001f, 9.9817633e-001f, 9.9817538e-001f,
9.9817443e-001f, 9.9817348e-001f, 9.9817252e-001f, 9.9817157e-001f,
9.9817062e-001f, 9.9816967e-001f, 9.9816872e-001f, 9.9816776e-001f,
9.9816681e-001f, 9.9816586e-001f, 9.9816491e-001f, 9.9816396e-001f,
9.9816301e-001f, 9.9816205e-001f, 9.9816110e-001f, 9.9816015e-001f,
9.9815920e-001f, 9.9815825e-001f, 9.9815729e-001f, 9.9815634e-001f,
9.9815539e-001f, 9.9815444e-001f, 9.9815349e-001f, 9.9815253e-001f,
9.9815158e-001f, 9.9815063e-001f, 9.9814968e-001f, 9.9814873e-001f,
9.9814777e-001f, 9.9814682e-001f, 9.9814587e-001f, 9.9814492e-001f,
9.9814397e-001f, 9.9814302e-001f, 9.9814206e-001f, 9.9814111e-001f,
9.9814016e-001f, 9.9813921e-001f, 9.9813826e-001f, 9.9813730e-001f,
9.9813635e-001f, 9.9813540e-001f, 9.9813445e-001f, 9.9813350e-001f,
9.9813254e-001f, 9.9813159e-001f, 9.9813064e-001f, 9.9812969e-001f,
9.9812874e-001f, 9.9812778e-001f, 9.9812683e-001f, 9.9812588e-001f,
9.9812493e-001f, 9.9812398e-001f, 9.9812303e-001f, 9.9812207e-001f,
9.9812112e-001f, 9.9812017e-001f, 9.9811922e-001f, 9.9811827e-001f,
9.9811731e-001f, 9.9811636e-001f, 9.9811541e-001f, 9.9811446e-001f,
9.9811351e-001f, 9.9811255e-001f, 9.9811160e-001f, 9.9811065e-001f,
9.9810970e-001f, 9.9810875e-001f, 9.9810780e-001f, 9.9810684e-001f,
9.9810589e-001f, 9.9810494e-001f, 9.9810399e-001f, 9.9810304e-001f,
9.9810208e-001f, 9.9810113e-001f, 9.9810018e-001f, 9.9809923e-001f,
9.9809828e-001f, 9.9809732e-001f, 9.9809637e-001f, 9.9809542e-001f,
9.9809447e-001f, 9.9809352e-001f, 9.9809257e-001f, 9.9809161e-001f,
9.9809066e-001f, 9.9808971e-001f, 9.9808876e-001f, 9.9808781e-001f,
9.9808685e-001f, 9.9808590e-001f, 9.9808495e-001f, 9.9808400e-001f,
9.9808305e-001f, 9.9808210e-001f, 9.9808114e-001f, 9.9808019e-001f,
9.9807924e-001f, 9.9807829e-001f, 9.9807734e-001f, 9.9807638e-001f,
9.9807543e-001f, 9.9807448e-001f, 9.9807353e-001f, 9.9807258e-001f,
9.9807162e-001f, 9.9807067e-001f, 9.9806972e-001f, 9.9806877e-001f,
9.9806782e-001f, 9.9806687e-001f, 9.9806591e-001f, 9.9806496e-001f,

```
    9.9806401e-001f, 9.9806306e-001f, 9.9806211e-001f, 9.9806115e-001f,
    9.9806020e-001f, 9.9805925e-001f, 9.9805830e-001f, 9.9805735e-001f,
    9.9805640e-001f, 9.9805544e-001f, 9.9805449e-001f, 9.9805354e-001f,
    9.9805259e-001f, 9.9805164e-001f, 9.9805068e-001f, 9.9804973e-001f

#else /*_MSC_VER && _M_IX86*/

    9.9999952e-001f, 9.9999857e-001f, 9.9999762e-001f, 9.9999666e-001f,
    9.9999571e-001f, 9.9999475e-001f, 9.9999380e-001f, 9.9999285e-001f,
    9.9999189e-001f, 9.9999094e-001f, 9.9998999e-001f, 9.9998903e-001f,
    9.9998808e-001f, 9.9998713e-001f, 9.9998617e-001f, 9.9998522e-001f,
    9.9998426e-001f, 9.9998331e-001f, 9.9998236e-001f, 9.9998140e-001f,
    9.9998045e-001f, 9.9997950e-001f, 9.9997854e-001f, 9.9997759e-001f,
    9.9997664e-001f, 9.9997568e-001f, 9.9997473e-001f, 9.9997377e-001f,
    9.9997282e-001f, 9.9997187e-001f, 9.9997091e-001f, 9.9996996e-001f,
    9.9996901e-001f, 9.9996805e-001f, 9.9996710e-001f, 9.9996615e-001f,
    9.9996519e-001f, 9.9996424e-001f, 9.9996328e-001f, 9.9996233e-001f,
    9.9996138e-001f, 9.9996042e-001f, 9.9995947e-001f, 9.9995852e-001f,
    9.9995756e-001f, 9.9995661e-001f, 9.9995566e-001f, 9.9995470e-001f,
    9.9995375e-001f, 9.9995279e-001f, 9.9995184e-001f, 9.9995089e-001f,
    9.9994993e-001f, 9.9994898e-001f, 9.9994803e-001f, 9.9994707e-001f,
    9.9994612e-001f, 9.9994517e-001f, 9.9994421e-001f, 9.9994326e-001f,
    9.9994230e-001f, 9.9994135e-001f, 9.9994040e-001f, 9.9993944e-001f,
    9.9993849e-001f, 9.9993754e-001f, 9.9993658e-001f, 9.9993563e-001f,
    9.9993468e-001f, 9.9993372e-001f, 9.9993277e-001f, 9.9993181e-001f,
    9.9993086e-001f, 9.9992991e-001f, 9.9992895e-001f, 9.9992800e-001f,
    9.9992705e-001f, 9.9992609e-001f, 9.9992514e-001f, 9.9992419e-001f,
    9.9992323e-001f, 9.9992228e-001f, 9.9992132e-001f, 9.9992037e-001f,
    9.9991942e-001f, 9.9991846e-001f, 9.9991751e-001f, 9.9991656e-001f,
    9.9991560e-001f, 9.9991465e-001f, 9.9991370e-001f, 9.9991274e-001f,
    9.9991179e-001f, 9.9991084e-001f, 9.9990988e-001f, 9.9990893e-001f,
    9.9990797e-001f, 9.9990702e-001f, 9.9990607e-001f, 9.9990511e-001f,
    9.9990416e-001f, 9.9990321e-001f, 9.9990225e-001f, 9.9990130e-001f,
    9.9990035e-001f, 9.9989939e-001f, 9.9989844e-001f, 9.9989749e-001f,
    9.9989653e-001f, 9.9989558e-001f, 9.9989462e-001f, 9.9989367e-001f,
    9.9989272e-001f, 9.9989176e-001f, 9.9989081e-001f, 9.9988986e-001f,
    9.9988890e-001f, 9.9988795e-001f, 9.9988700e-001f, 9.9988604e-001f,
    9.9988509e-001f, 9.9988414e-001f, 9.9988318e-001f, 9.9988223e-001f,
    9.9988127e-001f, 9.9988032e-001f, 9.9987937e-001f, 9.9987841e-001f,
    9.9987746e-001f, 9.9987651e-001f, 9.9987555e-001f, 9.9987460e-001f,
    9.9987365e-001f, 9.9987269e-001f, 9.9987174e-001f, 9.9987079e-001f,
    9.9986983e-001f, 9.9986888e-001f, 9.9986792e-001f, 9.9986697e-001f,
    9.9986602e-001f, 9.9986506e-001f, 9.9986411e-001f, 9.9986316e-001f,
    9.9986220e-001f, 9.9986125e-001f, 9.9986030e-001f, 9.9985934e-001f,
    9.9985839e-001f, 9.9985744e-001f, 9.9985648e-001f, 9.9985553e-001f,
    9.9985458e-001f, 9.9985362e-001f, 9.9985267e-001f, 9.9985171e-001f,
    9.9985076e-001f, 9.9984981e-001f, 9.9984885e-001f, 9.9984790e-001f,
    9.9984695e-001f, 9.9984599e-001f, 9.9984504e-001f, 9.9984409e-001f,
    9.9984313e-001f, 9.9984218e-001f, 9.9984123e-001f, 9.9984027e-001f,
    9.9983932e-001f, 9.9983837e-001f, 9.9983741e-001f, 9.9983646e-001f,
    9.9983550e-001f, 9.9983455e-001f, 9.9983360e-001f, 9.9983264e-001f,
    9.9983169e-001f, 9.9983074e-001f, 9.9982978e-001f, 9.9982883e-001f,
    9.9982788e-001f, 9.9982692e-001f, 9.9982597e-001f, 9.9982502e-001f,
    9.9982406e-001f, 9.9982311e-001f, 9.9982216e-001f, 9.9982120e-001f,
    9.9982025e-001f, 9.9981930e-001f, 9.9981834e-001f, 9.9981739e-001f,
    9.9981643e-001f, 9.9981548e-001f, 9.9981453e-001f, 9.9981357e-001f,
    9.9981262e-001f, 9.9981167e-001f, 9.9981071e-001f, 9.9980976e-001f,
    9.9980881e-001f, 9.9980785e-001f, 9.9980690e-001f, 9.9980595e-001f,
    9.9980499e-001f, 9.9980404e-001f, 9.9980309e-001f, 9.9980213e-001f,
    9.9980118e-001f, 9.9980023e-001f, 9.9979927e-001f, 9.9979832e-001f,
    9.9979736e-001f, 9.9979641e-001f, 9.9979546e-001f, 9.9979450e-001f,
    9.9979355e-001f, 9.9979260e-001f, 9.9979164e-001f, 9.9979069e-001f,
    9.9978974e-001f, 9.9978878e-001f, 9.9978783e-001f, 9.9978688e-001f,
    9.9978592e-001f, 9.9978497e-001f, 9.9978402e-001f, 9.9978306e-001f,
    9.9978211e-001f, 9.9978116e-001f, 9.9978020e-001f, 9.9977925e-001f,
    9.9977830e-001f, 9.9977734e-001f, 9.9977639e-001f, 9.9977543e-001f,
    9.9977448e-001f, 9.9977353e-001f, 9.9977257e-001f, 9.9977162e-001f,
    9.9977067e-001f, 9.9976971e-001f, 9.9976876e-001f, 9.9976781e-001f,
    9.9976685e-001f, 9.9976590e-001f, 9.9976495e-001f, 9.9976399e-001f,
    9.9976304e-001f, 9.9976209e-001f, 9.9976113e-001f, 9.9976018e-001f,
    9.9975923e-001f, 9.9975827e-001f, 9.9975732e-001f, 9.9975637e-001f,
    9.9975541e-001f, 9.9975446e-001f, 9.9975351e-001f, 9.9975255e-001f,
    9.9975160e-001f, 9.9975065e-001f, 9.9974969e-001f, 9.9974874e-001f,
    9.9974778e-001f, 9.9974683e-001f, 9.9974588e-001f, 9.9974492e-001f,
    9.9974397e-001f, 9.9974302e-001f, 9.9974206e-001f, 9.9974111e-001f,
    9.9974016e-001f, 9.9973920e-001f, 9.9973825e-001f, 9.9973730e-001f,
```

9.9973634e-001f, 9.9973539e-001f, 9.9973444e-001f, 9.9973348e-001f,
9.9973253e-001f, 9.9973158e-001f, 9.9973062e-001f, 9.9972967e-001f,
9.9972872e-001f, 9.9972776e-001f, 9.9972681e-001f, 9.9972586e-001f,
9.9972490e-001f, 9.9972395e-001f, 9.9972300e-001f, 9.9972204e-001f,
9.9972109e-001f, 9.9972014e-001f, 9.9971918e-001f, 9.9971823e-001f,
9.9971728e-001f, 9.9971632e-001f, 9.9971537e-001f, 9.9971442e-001f,
9.9971346e-001f, 9.9971251e-001f, 9.9971156e-001f, 9.9971060e-001f,
9.9970965e-001f, 9.9970869e-001f, 9.9970774e-001f, 9.9970679e-001f,
9.9970583e-001f, 9.9970488e-001f, 9.9970393e-001f, 9.9970297e-001f,
9.9970202e-001f, 9.9970107e-001f, 9.9970011e-001f, 9.9969916e-001f,
9.9969821e-001f, 9.9969725e-001f, 9.9969630e-001f, 9.9969535e-001f,
9.9969439e-001f, 9.9969344e-001f, 9.9969249e-001f, 9.9969153e-001f,
9.9969058e-001f, 9.9968963e-001f, 9.9968867e-001f, 9.9968772e-001f,
9.9968677e-001f, 9.9968581e-001f, 9.9968486e-001f, 9.9968391e-001f,
9.9968295e-001f, 9.9968200e-001f, 9.9968105e-001f, 9.9968009e-001f,
9.9967914e-001f, 9.9967819e-001f, 9.9967723e-001f, 9.9967628e-001f,
9.9967533e-001f, 9.9967437e-001f, 9.9967342e-001f, 9.9967247e-001f,
9.9967151e-001f, 9.9967056e-001f, 9.9966961e-001f, 9.9966865e-001f,
9.9966770e-001f, 9.9966675e-001f, 9.9966579e-001f, 9.9966484e-001f,
9.9966389e-001f, 9.9966293e-001f, 9.9966198e-001f, 9.9966103e-001f,
9.9966007e-001f, 9.9965912e-001f, 9.9965817e-001f, 9.9965721e-001f,
9.9965626e-001f, 9.9965531e-001f, 9.9965435e-001f, 9.9965340e-001f,
9.9965245e-001f, 9.9965149e-001f, 9.9965054e-001f, 9.9964959e-001f,
9.9964863e-001f, 9.9964768e-001f, 9.9964673e-001f, 9.9964577e-001f,
9.9964482e-001f, 9.9964387e-001f, 9.9964291e-001f, 9.9964196e-001f,
9.9964101e-001f, 9.9964005e-001f, 9.9963910e-001f, 9.9963815e-001f,
9.9963719e-001f, 9.9963624e-001f, 9.9963529e-001f, 9.9963433e-001f,
9.9963338e-001f, 9.9963243e-001f, 9.9963147e-001f, 9.9963052e-001f,
9.9962957e-001f, 9.9962861e-001f, 9.9962766e-001f, 9.9962671e-001f,
9.9962575e-001f, 9.9962480e-001f, 9.9962385e-001f, 9.9962289e-001f,
9.9962194e-001f, 9.9962099e-001f, 9.9962003e-001f, 9.9961908e-001f,
9.9961813e-001f, 9.9961717e-001f, 9.9961622e-001f, 9.9961527e-001f,
9.9961431e-001f, 9.9961336e-001f, 9.9961241e-001f, 9.9961145e-001f,
9.9961050e-001f, 9.9960955e-001f, 9.9960859e-001f, 9.9960764e-001f,
9.9960669e-001f, 9.9960573e-001f, 9.9960478e-001f, 9.9960383e-001f,
9.9960287e-001f, 9.9960192e-001f, 9.9960097e-001f, 9.9960001e-001f,
9.9959906e-001f, 9.9959811e-001f, 9.9959715e-001f, 9.9959620e-001f,
9.9959525e-001f, 9.9959429e-001f, 9.9959334e-001f, 9.9959239e-001f,
9.9959143e-001f, 9.9959048e-001f, 9.9958953e-001f, 9.9958857e-001f,
9.9958762e-001f, 9.9958667e-001f, 9.9958571e-001f, 9.9958476e-001f,
9.9958381e-001f, 9.9958285e-001f, 9.9958190e-001f, 9.9958095e-001f,
9.9957999e-001f, 9.9957904e-001f, 9.9957809e-001f, 9.9957713e-001f,
9.9957618e-001f, 9.9957523e-001f, 9.9957428e-001f, 9.9957332e-001f,
9.9957237e-001f, 9.9957142e-001f, 9.9957046e-001f, 9.9956951e-001f,
9.9956856e-001f, 9.9956760e-001f, 9.9956665e-001f, 9.9956570e-001f,
9.9956474e-001f, 9.9956379e-001f, 9.9956284e-001f, 9.9956188e-001f,
9.9956093e-001f, 9.9955998e-001f, 9.9955902e-001f, 9.9955807e-001f,
9.9955712e-001f, 9.9955616e-001f, 9.9955521e-001f, 9.9955426e-001f,
9.9955330e-001f, 9.9955235e-001f, 9.9955140e-001f, 9.9955044e-001f,
9.9954949e-001f, 9.9954854e-001f, 9.9954758e-001f, 9.9954663e-001f,
9.9954568e-001f, 9.9954472e-001f, 9.9954377e-001f, 9.9954282e-001f,
9.9954186e-001f, 9.9954091e-001f, 9.9953996e-001f, 9.9953900e-001f,
9.9953805e-001f, 9.9953710e-001f, 9.9953615e-001f, 9.9953519e-001f,
9.9953424e-001f, 9.9953329e-001f, 9.9953233e-001f, 9.9953138e-001f,
9.9953043e-001f, 9.9952947e-001f, 9.9952852e-001f, 9.9952757e-001f,
9.9952661e-001f, 9.9952566e-001f, 9.9952471e-001f, 9.9952375e-001f,
9.9952280e-001f, 9.9952185e-001f, 9.9952089e-001f, 9.9951994e-001f,
9.9951899e-001f, 9.9951803e-001f, 9.9951708e-001f, 9.9951613e-001f,
9.9951517e-001f, 9.9951422e-001f, 9.9951327e-001f, 9.9951231e-001f,
9.9951136e-001f, 9.9951041e-001f, 9.9950945e-001f, 9.9950850e-001f,
9.9950755e-001f, 9.9950660e-001f, 9.9950564e-001f, 9.9950469e-001f,
9.9950374e-001f, 9.9950278e-001f, 9.9950183e-001f, 9.9950088e-001f,
9.9949992e-001f, 9.9949897e-001f, 9.9949802e-001f, 9.9949706e-001f,
9.9949611e-001f, 9.9949516e-001f, 9.9949420e-001f, 9.9949325e-001f,
9.9949230e-001f, 9.9949134e-001f, 9.9949039e-001f, 9.9948944e-001f,
9.9948848e-001f, 9.9948753e-001f, 9.9948658e-001f, 9.9948563e-001f,
9.9948467e-001f, 9.9948372e-001f, 9.9948277e-001f, 9.9948181e-001f,
9.9948086e-001f, 9.9947991e-001f, 9.9947895e-001f, 9.9947800e-001f,
9.9947705e-001f, 9.9947609e-001f, 9.9947514e-001f, 9.9947419e-001f,
9.9947323e-001f, 9.9947228e-001f, 9.9947133e-001f, 9.9947037e-001f,
9.9946942e-001f, 9.9946847e-001f, 9.9946751e-001f, 9.9946656e-001f,
9.9946561e-001f, 9.9946466e-001f, 9.9946370e-001f, 9.9946275e-001f,
9.9946180e-001f, 9.9946084e-001f, 9.9945989e-001f, 9.9945894e-001f,
9.9945798e-001f, 9.9945703e-001f, 9.9945608e-001f, 9.9945512e-001f,
9.9945417e-001f, 9.9945322e-001f, 9.9945226e-001f, 9.9945131e-001f,
9.9945036e-001f, 9.9944940e-001f, 9.9944845e-001f, 9.9944750e-001f,

9.9944655e-001f, 9.9944559e-001f, 9.9944464e-001f, 9.9944369e-001f,
9.9944273e-001f, 9.9944178e-001f, 9.9944083e-001f, 9.9943987e-001f,
9.9943892e-001f, 9.9943797e-001f, 9.9943701e-001f, 9.9943606e-001f,
9.9943511e-001f, 9.9943415e-001f, 9.9943320e-001f, 9.9943225e-001f,
9.9943130e-001f, 9.9943034e-001f, 9.9942939e-001f, 9.9942844e-001f,
9.9942748e-001f, 9.9942653e-001f, 9.9942558e-001f, 9.9942462e-001f,
9.9942367e-001f, 9.9942272e-001f, 9.9942176e-001f, 9.9942081e-001f,
9.9941986e-001f, 9.9941890e-001f, 9.9941795e-001f, 9.9941700e-001f,
9.9941605e-001f, 9.9941509e-001f, 9.9941414e-001f, 9.9941319e-001f,
9.9941223e-001f, 9.9941128e-001f, 9.9941033e-001f, 9.9940937e-001f,
9.9940842e-001f, 9.9940747e-001f, 9.9940651e-001f, 9.9940556e-001f,
9.9940461e-001f, 9.9940365e-001f, 9.9940270e-001f, 9.9940175e-001f,
9.9940080e-001f, 9.9939984e-001f, 9.9939889e-001f, 9.9939794e-001f,
9.9939698e-001f, 9.9939603e-001f, 9.9939508e-001f, 9.9939412e-001f,
9.9939317e-001f, 9.9939222e-001f, 9.9939126e-001f, 9.9939031e-001f,
9.9938936e-001f, 9.9938841e-001f, 9.9938745e-001f, 9.9938650e-001f,
9.9938555e-001f, 9.9938459e-001f, 9.9938364e-001f, 9.9938269e-001f,
9.9938173e-001f, 9.9938078e-001f, 9.9937983e-001f, 9.9937887e-001f,
9.9937792e-001f, 9.9937697e-001f, 9.9937601e-001f, 9.9937506e-001f,
9.9937411e-001f, 9.9937316e-001f, 9.9937220e-001f, 9.9937125e-001f,
9.9937030e-001f, 9.9936934e-001f, 9.9936839e-001f, 9.9936744e-001f,
9.9936648e-001f, 9.9936553e-001f, 9.9936458e-001f, 9.9936362e-001f,
9.9936267e-001f, 9.9936172e-001f, 9.9936077e-001f, 9.9935981e-001f,
9.9935886e-001f, 9.9935791e-001f, 9.9935695e-001f, 9.9935600e-001f,
9.9935505e-001f, 9.9935409e-001f, 9.9935314e-001f, 9.9935219e-001f,
9.9935124e-001f, 9.9935028e-001f, 9.9934933e-001f, 9.9934838e-001f,
9.9934742e-001f, 9.9934647e-001f, 9.9934552e-001f, 9.9934456e-001f,
9.9934361e-001f, 9.9934266e-001f, 9.9934170e-001f, 9.9934075e-001f,
9.9933980e-001f, 9.9933885e-001f, 9.9933789e-001f, 9.9933694e-001f,
9.9933599e-001f, 9.9933503e-001f, 9.9933408e-001f, 9.9933313e-001f,
9.9933217e-001f, 9.9933122e-001f, 9.9933027e-001f, 9.9932932e-001f,
9.9932836e-001f, 9.9932741e-001f, 9.9932646e-001f, 9.9932550e-001f,
9.9932455e-001f, 9.9932360e-001f, 9.9932264e-001f, 9.9932169e-001f,
9.9932074e-001f, 9.9931978e-001f, 9.9931883e-001f, 9.9931788e-001f,
9.9931693e-001f, 9.9931597e-001f, 9.9931502e-001f, 9.9931407e-001f,
9.9931311e-001f, 9.9931216e-001f, 9.9931121e-001f, 9.9931025e-001f,
9.9930930e-001f, 9.9930835e-001f, 9.9930740e-001f, 9.9930644e-001f,
9.9930549e-001f, 9.9930454e-001f, 9.9930358e-001f, 9.9930263e-001f,
9.9930168e-001f, 9.9930072e-001f, 9.9929977e-001f, 9.9929882e-001f,
9.9929787e-001f, 9.9929691e-001f, 9.9929596e-001f, 9.9929501e-001f,
9.9929405e-001f, 9.9929310e-001f, 9.9929215e-001f, 9.9929119e-001f,
9.9929024e-001f, 9.9928929e-001f, 9.9928834e-001f, 9.9928738e-001f,
9.9928643e-001f, 9.9928548e-001f, 9.9928452e-001f, 9.9928357e-001f,
9.9928262e-001f, 9.9928166e-001f, 9.9928071e-001f, 9.9927976e-001f,
9.9927881e-001f, 9.9927785e-001f, 9.9927690e-001f, 9.9927595e-001f,
9.9927499e-001f, 9.9927404e-001f, 9.9927309e-001f, 9.9927213e-001f,
9.9927118e-001f, 9.9927023e-001f, 9.9926928e-001f, 9.9926832e-001f,
9.9926737e-001f, 9.9926642e-001f, 9.9926546e-001f, 9.9926451e-001f,
9.9926356e-001f, 9.9926260e-001f, 9.9926165e-001f, 9.9926070e-001f,
9.9925975e-001f, 9.9925879e-001f, 9.9925784e-001f, 9.9925689e-001f,
9.9925593e-001f, 9.9925498e-001f, 9.9925403e-001f, 9.9925308e-001f,
9.9925212e-001f, 9.9925117e-001f, 9.9925022e-001f, 9.9924926e-001f,
9.9924831e-001f, 9.9924736e-001f, 9.9924640e-001f, 9.9924545e-001f,
9.9924450e-001f, 9.9924355e-001f, 9.9924259e-001f, 9.9924164e-001f,
9.9924069e-001f, 9.9923973e-001f, 9.9923878e-001f, 9.9923783e-001f,
9.9923688e-001f, 9.9923592e-001f, 9.9923497e-001f, 9.9923402e-001f,
9.9923306e-001f, 9.9923211e-001f, 9.9923116e-001f, 9.9923020e-001f,
9.9922925e-001f, 9.9922830e-001f, 9.9922735e-001f, 9.9922639e-001f,
9.9922544e-001f, 9.9922449e-001f, 9.9922353e-001f, 9.9922258e-001f,
9.9922163e-001f, 9.9922068e-001f, 9.9921972e-001f, 9.9921877e-001f,
9.9921782e-001f, 9.9921686e-001f, 9.9921591e-001f, 9.9921496e-001f,
9.9921400e-001f, 9.9921305e-001f, 9.9921210e-001f, 9.9921115e-001f,
9.9921019e-001f, 9.9920924e-001f, 9.9920829e-001f, 9.9920733e-001f,
9.9920638e-001f, 9.9920543e-001f, 9.9920448e-001f, 9.9920352e-001f,
9.9920257e-001f, 9.9920162e-001f, 9.9920066e-001f, 9.9919971e-001f,
9.9919876e-001f, 9.9919780e-001f, 9.9919685e-001f, 9.9919590e-001f,
9.9919495e-001f, 9.9919399e-001f, 9.9919304e-001f, 9.9919209e-001f,
9.9919113e-001f, 9.9919018e-001f, 9.9918923e-001f, 9.9918828e-001f,
9.9918732e-001f, 9.9918637e-001f, 9.9918542e-001f, 9.9918446e-001f,
9.9918351e-001f, 9.9918256e-001f, 9.9918161e-001f, 9.9918065e-001f,
9.9917970e-001f, 9.9917875e-001f, 9.9917779e-001f, 9.9917684e-001f,
9.9917589e-001f, 9.9917494e-001f, 9.9917398e-001f, 9.9917303e-001f,
9.9917208e-001f, 9.9917112e-001f, 9.9917017e-001f, 9.9916922e-001f,
9.9916827e-001f, 9.9916731e-001f, 9.9916636e-001f, 9.9916541e-001f,
9.9916445e-001f, 9.9916350e-001f, 9.9916255e-001f, 9.9916160e-001f,
9.9916064e-001f, 9.9915969e-001f, 9.9915874e-001f, 9.9915778e-001f,

```
9.9915683e-001f, 9.9915588e-001f, 9.9915492e-001f, 9.9915397e-001f,
9.9915302e-001f, 9.9915207e-001f, 9.9915111e-001f, 9.9915016e-001f,
9.9914921e-001f, 9.9914825e-001f, 9.9914730e-001f, 9.9914635e-001f,
9.9914540e-001f, 9.9914444e-001f, 9.9914349e-001f, 9.9914254e-001f,
9.9914158e-001f, 9.9914063e-001f, 9.9913968e-001f, 9.9913873e-001f,
9.9913777e-001f, 9.9913682e-001f, 9.9913587e-001f, 9.9913491e-001f,
9.9913396e-001f, 9.9913301e-001f, 9.9913206e-001f, 9.9913110e-001f,
9.9913015e-001f, 9.9912920e-001f, 9.9912825e-001f, 9.9912729e-001f,
9.9912634e-001f, 9.9912539e-001f, 9.9912443e-001f, 9.9912348e-001f,
9.9912253e-001f, 9.9912158e-001f, 9.9912062e-001f, 9.9911967e-001f,
9.9911872e-001f, 9.9911776e-001f, 9.9911681e-001f, 9.9911586e-001f,
9.9911491e-001f, 9.9911395e-001f, 9.9911300e-001f, 9.9911205e-001f,
9.9911109e-001f, 9.9911014e-001f, 9.9910919e-001f, 9.9910824e-001f,
9.9910728e-001f, 9.9910633e-001f, 9.9910538e-001f, 9.9910442e-001f,
9.9910347e-001f, 9.9910252e-001f, 9.9910157e-001f, 9.9910061e-001f,
9.9909966e-001f, 9.9909871e-001f, 9.9909775e-001f, 9.9909680e-001f,
9.9909585e-001f, 9.9909490e-001f, 9.9909394e-001f, 9.9909299e-001f,
9.9909204e-001f, 9.9909108e-001f, 9.9909013e-001f, 9.9908918e-001f,
9.9908823e-001f, 9.9908727e-001f, 9.9908632e-001f, 9.9908537e-001f,
9.9908442e-001f, 9.9908346e-001f, 9.9908251e-001f, 9.9908156e-001f,
9.9908060e-001f, 9.9907965e-001f, 9.9907870e-001f, 9.9907775e-001f,
9.9907679e-001f, 9.9907584e-001f, 9.9907489e-001f, 9.9907393e-001f,
9.9907298e-001f, 9.9907203e-001f, 9.9907108e-001f, 9.9907012e-001f,
9.9906917e-001f, 9.9906822e-001f, 9.9906726e-001f, 9.9906631e-001f,
9.9906536e-001f, 9.9906441e-001f, 9.9906345e-001f, 9.9906250e-001f,
9.9906155e-001f, 9.9906060e-001f, 9.9905964e-001f, 9.9905869e-001f,
9.9905774e-001f, 9.9905678e-001f, 9.9905583e-001f, 9.9905488e-001f,
9.9905393e-001f, 9.9905297e-001f, 9.9905202e-001f, 9.9905107e-001f,
9.9905011e-001f, 9.9904916e-001f, 9.9904821e-001f, 9.9904726e-001f,
9.9904630e-001f, 9.9904535e-001f, 9.9904440e-001f, 9.9904345e-001f,
9.9904249e-001f, 9.9904154e-001f, 9.9904059e-001f, 9.9903963e-001f,
9.9903868e-001f, 9.9903773e-001f, 9.9903678e-001f, 9.9903582e-001f,
9.9903487e-001f, 9.9903392e-001f, 9.9903297e-001f, 9.9903201e-001f,
9.9903106e-001f, 9.9903011e-001f, 9.9902915e-001f, 9.9902820e-001f,
9.9902725e-001f, 9.9902630e-001f, 9.9902534e-001f, 9.9902439e-001f,
9.9902344e-001f, 9.9902249e-001f, 9.9902153e-001f, 9.9902058e-001f,
9.9901963e-001f, 9.9901867e-001f, 9.9901772e-001f, 9.9901677e-001f,
9.9901582e-001f, 9.9901486e-001f, 9.9901391e-001f, 9.9901296e-001f,
9.9901200e-001f, 9.9901105e-001f, 9.9901010e-001f, 9.9900915e-001f,
9.9900819e-001f, 9.9900724e-001f, 9.9900629e-001f, 9.9900534e-001f,
9.9900438e-001f, 9.9900343e-001f, 9.9900248e-001f, 9.9900152e-001f,
9.9900057e-001f, 9.9899962e-001f, 9.9899867e-001f, 9.9899771e-001f,
9.9899676e-001f, 9.9899581e-001f, 9.9899486e-001f, 9.9899390e-001f,
9.9899295e-001f, 9.9899200e-001f, 9.9899105e-001f, 9.9899009e-001f,
9.9898914e-001f, 9.9898819e-001f, 9.9898723e-001f, 9.9898628e-001f,
9.9898533e-001f, 9.9898438e-001f, 9.9898342e-001f, 9.9898247e-001f,
9.9898152e-001f, 9.9898057e-001f, 9.9897961e-001f, 9.9897866e-001f,
9.9897771e-001f, 9.9897675e-001f, 9.9897580e-001f, 9.9897485e-001f,
9.9897390e-001f, 9.9897294e-001f, 9.9897199e-001f, 9.9897104e-001f,
9.9897009e-001f, 9.9896913e-001f, 9.9896818e-001f, 9.9896723e-001f,
9.9896627e-001f, 9.9896532e-001f, 9.9896437e-001f, 9.9896342e-001f,
9.9896246e-001f, 9.9896151e-001f, 9.9896056e-001f, 9.9895961e-001f,
9.9895865e-001f, 9.9895770e-001f, 9.9895675e-001f, 9.9895580e-001f,
9.9895484e-001f, 9.9895389e-001f, 9.9895294e-001f, 9.9895198e-001f,
9.9895103e-001f, 9.9895008e-001f, 9.9894913e-001f, 9.9894817e-001f,
9.9894722e-001f, 9.9894627e-001f, 9.9894532e-001f, 9.9894436e-001f,
9.9894341e-001f, 9.9894246e-001f, 9.9894151e-001f, 9.9894055e-001f,
9.9893960e-001f, 9.9893865e-001f, 9.9893769e-001f, 9.9893674e-001f,
9.9893579e-001f, 9.9893484e-001f, 9.9893388e-001f, 9.9893293e-001f,
9.9893198e-001f, 9.9893103e-001f, 9.9893007e-001f, 9.9892912e-001f,
9.9892817e-001f, 9.9892722e-001f, 9.9892626e-001f, 9.9892531e-001f,
9.9892436e-001f, 9.9892340e-001f, 9.9892245e-001f, 9.9892150e-001f,
9.9892055e-001f, 9.9891959e-001f, 9.9891864e-001f, 9.9891769e-001f,
9.9891674e-001f, 9.9891578e-001f, 9.9891483e-001f, 9.9891388e-001f,
9.9891293e-001f, 9.9891197e-001f, 9.9891102e-001f, 9.9891007e-001f,
9.9890912e-001f, 9.9890816e-001f, 9.9890721e-001f, 9.9890626e-001f,
9.9890530e-001f, 9.9890435e-001f, 9.9890340e-001f, 9.9890245e-001f,
9.9890149e-001f, 9.9890054e-001f, 9.9889959e-001f, 9.9889864e-001f,
9.9889768e-001f, 9.9889673e-001f, 9.9889578e-001f, 9.9889483e-001f,
9.9889387e-001f, 9.9889292e-001f, 9.9889197e-001f, 9.9889102e-001f,
9.9889006e-001f, 9.9888911e-001f, 9.9888816e-001f, 9.9888720e-001f,
9.9888625e-001f, 9.9888530e-001f, 9.9888435e-001f, 9.9888339e-001f,
9.9888244e-001f, 9.9888149e-001f, 9.9888054e-001f, 9.9887958e-001f,
9.9887863e-001f, 9.9887768e-001f, 9.9887673e-001f, 9.9887577e-001f,
9.9887482e-001f, 9.9887387e-001f, 9.9887292e-001f, 9.9887196e-001f,
9.9887101e-001f, 9.9887006e-001f, 9.9886911e-001f, 9.9886815e-001f,
```

9.9886720e-001f, 9.9886625e-001f, 9.9886529e-001f, 9.9886434e-001f,
9.9886339e-001f, 9.9886244e-001f, 9.9886148e-001f, 9.9886053e-001f,
9.9885958e-001f, 9.9885863e-001f, 9.9885767e-001f, 9.9885672e-001f,
9.9885577e-001f, 9.9885482e-001f, 9.9885386e-001f, 9.9885291e-001f,
9.9885196e-001f, 9.9885101e-001f, 9.9885005e-001f, 9.9884910e-001f,
9.9884815e-001f, 9.9884720e-001f, 9.9884624e-001f, 9.9884529e-001f,
9.9884434e-001f, 9.9884339e-001f, 9.9884243e-001f, 9.9884148e-001f,
9.9884053e-001f, 9.9883958e-001f, 9.9883862e-001f, 9.9883767e-001f,
9.9883672e-001f, 9.9883577e-001f, 9.9883481e-001f, 9.9883386e-001f,
9.9883291e-001f, 9.9883195e-001f, 9.9883100e-001f, 9.9883005e-001f,
9.9882910e-001f, 9.9882814e-001f, 9.9882719e-001f, 9.9882624e-001f,
9.9882529e-001f, 9.9882433e-001f, 9.9882338e-001f, 9.9882243e-001f,
9.9882148e-001f, 9.9882052e-001f, 9.9881957e-001f, 9.9881862e-001f,
9.9881767e-001f, 9.9881671e-001f, 9.9881576e-001f, 9.9881481e-001f,
9.9881386e-001f, 9.9881290e-001f, 9.9881195e-001f, 9.9881100e-001f,
9.9881005e-001f, 9.9880909e-001f, 9.9880814e-001f, 9.9880719e-001f,
9.9880624e-001f, 9.9880528e-001f, 9.9880433e-001f, 9.9880338e-001f,
9.9880243e-001f, 9.9880147e-001f, 9.9880052e-001f, 9.9879957e-001f,
9.9879862e-001f, 9.9879766e-001f, 9.9879671e-001f, 9.9879576e-001f,
9.9879481e-001f, 9.9879385e-001f, 9.9879290e-001f, 9.9879195e-001f,
9.9879100e-001f, 9.9879004e-001f, 9.9878909e-001f, 9.9878814e-001f,
9.9878719e-001f, 9.9878623e-001f, 9.9878528e-001f, 9.9878433e-001f,
9.9878338e-001f, 9.9878242e-001f, 9.9878147e-001f, 9.9878052e-001f,
9.9877957e-001f, 9.9877861e-001f, 9.9877766e-001f, 9.9877671e-001f,
9.9877576e-001f, 9.9877480e-001f, 9.9877385e-001f, 9.9877290e-001f,
9.9877195e-001f, 9.9877099e-001f, 9.9877004e-001f, 9.9876909e-001f,
9.9876814e-001f, 9.9876718e-001f, 9.9876623e-001f, 9.9876528e-001f,
9.9876433e-001f, 9.9876337e-001f, 9.9876242e-001f, 9.9876147e-001f,
9.9876052e-001f, 9.9875956e-001f, 9.9875861e-001f, 9.9875766e-001f,
9.9875671e-001f, 9.9875575e-001f, 9.9875480e-001f, 9.9875385e-001f,
9.9875290e-001f, 9.9875194e-001f, 9.9875099e-001f, 9.9875004e-001f,
9.9874909e-001f, 9.9874813e-001f, 9.9874718e-001f, 9.9874623e-001f,
9.9874528e-001f, 9.9874432e-001f, 9.9874337e-001f, 9.9874242e-001f,
9.9874147e-001f, 9.9874051e-001f, 9.9873956e-001f, 9.9873861e-001f,
9.9873766e-001f, 9.9873670e-001f, 9.9873575e-001f, 9.9873480e-001f,
9.9873385e-001f, 9.9873289e-001f, 9.9873194e-001f, 9.9873099e-001f,
9.9873004e-001f, 9.9872908e-001f, 9.9872813e-001f, 9.9872718e-001f,
9.9872623e-001f, 9.9872527e-001f, 9.9872432e-001f, 9.9872337e-001f,
9.9872242e-001f, 9.9872146e-001f, 9.9872051e-001f, 9.9871956e-001f,
9.9871861e-001f, 9.9871765e-001f, 9.9871670e-001f, 9.9871575e-001f,
9.9871480e-001f, 9.9871384e-001f, 9.9871289e-001f, 9.9871194e-001f,
9.9871099e-001f, 9.9871003e-001f, 9.9870908e-001f, 9.9870813e-001f,
9.9870718e-001f, 9.9870622e-001f, 9.9870527e-001f, 9.9870432e-001f,
9.9870337e-001f, 9.9870242e-001f, 9.9870146e-001f, 9.9870051e-001f,
9.9869956e-001f, 9.9869861e-001f, 9.9869765e-001f, 9.9869670e-001f,
9.9869575e-001f, 9.9869480e-001f, 9.9869384e-001f, 9.9869289e-001f,
9.9869194e-001f, 9.9869099e-001f, 9.9869003e-001f, 9.9868908e-001f,
9.9868813e-001f, 9.9868718e-001f, 9.9868622e-001f, 9.9868527e-001f,
9.9868432e-001f, 9.9868337e-001f, 9.9868241e-001f, 9.9868146e-001f,
9.9868051e-001f, 9.9867956e-001f, 9.9867860e-001f, 9.9867765e-001f,
9.9867670e-001f, 9.9867575e-001f, 9.9867479e-001f, 9.9867384e-001f,
9.9867289e-001f, 9.9867194e-001f, 9.9867099e-001f, 9.9867003e-001f,
9.9866908e-001f, 9.9866813e-001f, 9.9866718e-001f, 9.9866622e-001f,
9.9866527e-001f, 9.9866432e-001f, 9.9866337e-001f, 9.9866241e-001f,
9.9866146e-001f, 9.9866051e-001f, 9.9865956e-001f, 9.9865860e-001f,
9.9865765e-001f, 9.9865670e-001f, 9.9865575e-001f, 9.9865479e-001f,
9.9865384e-001f, 9.9865289e-001f, 9.9865194e-001f, 9.9865098e-001f,
9.9865003e-001f, 9.9864908e-001f, 9.9864813e-001f, 9.9864718e-001f,
9.9864622e-001f, 9.9864527e-001f, 9.9864432e-001f, 9.9864337e-001f,
9.9864241e-001f, 9.9864146e-001f, 9.9864051e-001f, 9.9863956e-001f,
9.9863860e-001f, 9.9863765e-001f, 9.9863670e-001f, 9.9863575e-001f,
9.9863479e-001f, 9.9863384e-001f, 9.9863289e-001f, 9.9863194e-001f,
9.9863098e-001f, 9.9863003e-001f, 9.9862908e-001f, 9.9862813e-001f,
9.9862718e-001f, 9.9862622e-001f, 9.9862527e-001f, 9.9862432e-001f,
9.9862337e-001f, 9.9862241e-001f, 9.9862146e-001f, 9.9862051e-001f,
9.9861956e-001f, 9.9861860e-001f, 9.9861765e-001f, 9.9861670e-001f,
9.9861575e-001f, 9.9861479e-001f, 9.9861384e-001f, 9.9861289e-001f,
9.9861194e-001f, 9.9861099e-001f, 9.9861003e-001f, 9.9860908e-001f,
9.9860813e-001f, 9.9860718e-001f, 9.9860622e-001f, 9.9860527e-001f,
9.9860432e-001f, 9.9860337e-001f, 9.9860241e-001f, 9.9860146e-001f,
9.9860051e-001f, 9.9859956e-001f, 9.9859860e-001f, 9.9859765e-001f,
9.9859670e-001f, 9.9859575e-001f, 9.9859480e-001f, 9.9859384e-001f,
9.9859289e-001f, 9.9859194e-001f, 9.9859099e-001f, 9.9859003e-001f,
9.9858908e-001f, 9.9858813e-001f, 9.9858718e-001f, 9.9858622e-001f,
9.9858527e-001f, 9.9858432e-001f, 9.9858337e-001f, 9.9858242e-001f,
9.9858146e-001f, 9.9858051e-001f, 9.9857956e-001f, 9.9857861e-001f,

9.9857765e-001f, 9.9857670e-001f, 9.9857575e-001f, 9.9857480e-001f,
9.9857384e-001f, 9.9857289e-001f, 9.9857194e-001f, 9.9857099e-001f,
9.9857004e-001f, 9.9856908e-001f, 9.9856813e-001f, 9.9856718e-001f,
9.9856623e-001f, 9.9856527e-001f, 9.9856432e-001f, 9.9856337e-001f,
9.9856242e-001f, 9.9856146e-001f, 9.9856051e-001f, 9.9855956e-001f,
9.9855861e-001f, 9.9855766e-001f, 9.9855670e-001f, 9.9855575e-001f,
9.9855480e-001f, 9.9855385e-001f, 9.9855289e-001f, 9.9855194e-001f,
9.9855099e-001f, 9.9855004e-001f, 9.9854908e-001f, 9.9854813e-001f,
9.9854718e-001f, 9.9854623e-001f, 9.9854528e-001f, 9.9854432e-001f,
9.9854337e-001f, 9.9854242e-001f, 9.9854147e-001f, 9.9854051e-001f,
9.9853956e-001f, 9.9853861e-001f, 9.9853766e-001f, 9.9853670e-001f,
9.9853575e-001f, 9.9853480e-001f, 9.9853385e-001f, 9.9853290e-001f,
9.9853194e-001f, 9.9853099e-001f, 9.9853004e-001f, 9.9852909e-001f,
9.9852813e-001f, 9.9852718e-001f, 9.9852623e-001f, 9.9852528e-001f,
9.9852433e-001f, 9.9852337e-001f, 9.9852242e-001f, 9.9852147e-001f,
9.9852052e-001f, 9.9851956e-001f, 9.9851861e-001f, 9.9851766e-001f,
9.9851671e-001f, 9.9851575e-001f, 9.9851480e-001f, 9.9851385e-001f,
9.9851290e-001f, 9.9851195e-001f, 9.9851099e-001f, 9.9851004e-001f,
9.9850909e-001f, 9.9850814e-001f, 9.9850718e-001f, 9.9850623e-001f,
9.9850528e-001f, 9.9850433e-001f, 9.9850338e-001f, 9.9850242e-001f,
9.9850147e-001f, 9.9850052e-001f, 9.9849957e-001f, 9.9849861e-001f,
9.9849766e-001f, 9.9849671e-001f, 9.9849576e-001f, 9.9849481e-001f,
9.9849385e-001f, 9.9849290e-001f, 9.9849195e-001f, 9.9849100e-001f,
9.9849004e-001f, 9.9848909e-001f, 9.9848814e-001f, 9.9848719e-001f,
9.9848624e-001f, 9.9848528e-001f, 9.9848433e-001f, 9.9848338e-001f,
9.9848243e-001f, 9.9848147e-001f, 9.9848052e-001f, 9.9847957e-001f,
9.9847862e-001f, 9.9847767e-001f, 9.9847671e-001f, 9.9847576e-001f,
9.9847481e-001f, 9.9847386e-001f, 9.9847290e-001f, 9.9847195e-001f,
9.9847100e-001f, 9.9847005e-001f, 9.9846910e-001f, 9.9846814e-001f,
9.9846719e-001f, 9.9846624e-001f, 9.9846529e-001f, 9.9846433e-001f,
9.9846338e-001f, 9.9846243e-001f, 9.9846148e-001f, 9.9846053e-001f,
9.9845957e-001f, 9.9845862e-001f, 9.9845767e-001f, 9.9845672e-001f,
9.9845576e-001f, 9.9845481e-001f, 9.9845386e-001f, 9.9845291e-001f,
9.9845196e-001f, 9.9845100e-001f, 9.9845005e-001f, 9.9844910e-001f,
9.9844815e-001f, 9.9844719e-001f, 9.9844624e-001f, 9.9844529e-001f,
9.9844434e-001f, 9.9844339e-001f, 9.9844243e-001f, 9.9844148e-001f,
9.9844053e-001f, 9.9843958e-001f, 9.9843862e-001f, 9.9843767e-001f,
9.9843672e-001f, 9.9843577e-001f, 9.9843482e-001f, 9.9843386e-001f,
9.9843291e-001f, 9.9843196e-001f, 9.9843101e-001f, 9.9843006e-001f,
9.9842910e-001f, 9.9842815e-001f, 9.9842720e-001f, 9.9842625e-001f,
9.9842529e-001f, 9.9842434e-001f, 9.9842339e-001f, 9.9842244e-001f,
9.9842149e-001f, 9.9842053e-001f, 9.9841958e-001f, 9.9841863e-001f,
9.9841768e-001f, 9.9841672e-001f, 9.9841577e-001f, 9.9841482e-001f,
9.9841387e-001f, 9.9841292e-001f, 9.9841196e-001f, 9.9841101e-001f,
9.9841006e-001f, 9.9840911e-001f, 9.9840816e-001f, 9.9840720e-001f,
9.9840625e-001f, 9.9840530e-001f, 9.9840435e-001f, 9.9840339e-001f,
9.9840244e-001f, 9.9840149e-001f, 9.9840054e-001f, 9.9839959e-001f,
9.9839863e-001f, 9.9839768e-001f, 9.9839673e-001f, 9.9839578e-001f,
9.9839483e-001f, 9.9839387e-001f, 9.9839292e-001f, 9.9839197e-001f,
9.9839102e-001f, 9.9839006e-001f, 9.9838911e-001f, 9.9838816e-001f,
9.9838721e-001f, 9.9838626e-001f, 9.9838530e-001f, 9.9838435e-001f,
9.9838340e-001f, 9.9838245e-001f, 9.9838150e-001f, 9.9838054e-001f,
9.9837959e-001f, 9.9837864e-001f, 9.9837769e-001f, 9.9837673e-001f,
9.9837578e-001f, 9.9837483e-001f, 9.9837388e-001f, 9.9837293e-001f,
9.9837197e-001f, 9.9837102e-001f, 9.9837007e-001f, 9.9836912e-001f,
9.9836817e-001f, 9.9836721e-001f, 9.9836626e-001f, 9.9836531e-001f,
9.9836436e-001f, 9.9836341e-001f, 9.9836245e-001f, 9.9836150e-001f,
9.9836055e-001f, 9.9835960e-001f, 9.9835864e-001f, 9.9835769e-001f,
9.9835674e-001f, 9.9835579e-001f, 9.9835484e-001f, 9.9835388e-001f,
9.9835293e-001f, 9.9835198e-001f, 9.9835103e-001f, 9.9835008e-001f,
9.9834912e-001f, 9.9834817e-001f, 9.9834722e-001f, 9.9834627e-001f,
9.9834532e-001f, 9.9834436e-001f, 9.9834341e-001f, 9.9834246e-001f,
9.9834151e-001f, 9.9834055e-001f, 9.9833960e-001f, 9.9833865e-001f,
9.9833770e-001f, 9.9833675e-001f, 9.9833579e-001f, 9.9833484e-001f,
9.9833389e-001f, 9.9833294e-001f, 9.9833199e-001f, 9.9833103e-001f,
9.9833008e-001f, 9.9832913e-001f, 9.9832818e-001f, 9.9832723e-001f,
9.9832627e-001f, 9.9832532e-001f, 9.9832437e-001f, 9.9832342e-001f,
9.9832247e-001f, 9.9832151e-001f, 9.9832056e-001f, 9.9831961e-001f,
9.9831866e-001f, 9.9831770e-001f, 9.9831675e-001f, 9.9831580e-001f,
9.9831485e-001f, 9.9831390e-001f, 9.9831294e-001f, 9.9831199e-001f,
9.9831104e-001f, 9.9831009e-001f, 9.9830914e-001f, 9.9830818e-001f,
9.9830723e-001f, 9.9830628e-001f, 9.9830533e-001f, 9.9830438e-001f,
9.9830342e-001f, 9.9830247e-001f, 9.9830152e-001f, 9.9830057e-001f,
9.9829962e-001f, 9.9829866e-001f, 9.9829771e-001f, 9.9829676e-001f,
9.9829581e-001f, 9.9829486e-001f, 9.9829390e-001f, 9.9829295e-001f,
9.9829200e-001f, 9.9829105e-001f, 9.9829009e-001f, 9.9828914e-001f,

```
    9.9828819e-001f, 9.9828724e-001f, 9.9828629e-001f, 9.9828533e-001f,
    9.9828438e-001f, 9.9828343e-001f, 9.9828248e-001f, 9.9828153e-001f,
    9.9828057e-001f, 9.9827962e-001f, 9.9827867e-001f, 9.9827772e-001f,
    9.9827677e-001f, 9.9827581e-001f, 9.9827486e-001f, 9.9827391e-001f,
    9.9827296e-001f, 9.9827201e-001f, 9.9827105e-001f, 9.9827010e-001f,
    9.9826915e-001f, 9.9826820e-001f, 9.9826725e-001f, 9.9826629e-001f,
    9.9826534e-001f, 9.9826439e-001f, 9.9826344e-001f, 9.9826249e-001f,
    9.9826153e-001f, 9.9826058e-001f, 9.9825963e-001f, 9.9825868e-001f,
    9.9825773e-001f, 9.9825677e-001f, 9.9825582e-001f, 9.9825487e-001f,
    9.9825392e-001f, 9.9825297e-001f, 9.9825201e-001f, 9.9825106e-001f,
    9.9825011e-001f, 9.9824916e-001f, 9.9824821e-001f, 9.9824725e-001f,
    9.9824630e-001f, 9.9824535e-001f, 9.9824440e-001f, 9.9824345e-001f,
    9.9824249e-001f, 9.9824154e-001f, 9.9824059e-001f, 9.9823964e-001f,
    9.9823869e-001f, 9.9823773e-001f, 9.9823678e-001f, 9.9823583e-001f,
    9.9823488e-001f, 9.9823393e-001f, 9.9823297e-001f, 9.9823202e-001f,
    9.9823107e-001f, 9.9823012e-001f, 9.9822917e-001f, 9.9822821e-001f,
    9.9822726e-001f, 9.9822631e-001f, 9.9822536e-001f, 9.9822441e-001f,
    9.9822345e-001f, 9.9822250e-001f, 9.9822155e-001f, 9.9822060e-001f,
    9.9821965e-001f, 9.9821869e-001f, 9.9821774e-001f, 9.9821679e-001f,
    9.9821584e-001f, 9.9821489e-001f, 9.9821393e-001f, 9.9821298e-001f,
    9.9821203e-001f, 9.9821108e-001f, 9.9821013e-001f, 9.9820917e-001f,
    9.9820822e-001f, 9.9820727e-001f, 9.9820632e-001f, 9.9820537e-001f,
    9.9820441e-001f, 9.9820346e-001f, 9.9820251e-001f, 9.9820156e-001f,
    9.9820061e-001f, 9.9819965e-001f, 9.9819870e-001f, 9.9819775e-001f,
    9.9819680e-001f, 9.9819585e-001f, 9.9819490e-001f, 9.9819394e-001f,
    9.9819299e-001f, 9.9819204e-001f, 9.9819109e-001f, 9.9819014e-001f,
    9.9818918e-001f, 9.9818823e-001f, 9.9818728e-001f, 9.9818633e-001f,
    9.9818538e-001f, 9.9818442e-001f, 9.9818347e-001f, 9.9818252e-001f,
    9.9818157e-001f, 9.9818062e-001f, 9.9817966e-001f, 9.9817871e-001f,
    9.9817776e-001f, 9.9817681e-001f, 9.9817586e-001f, 9.9817490e-001f,
    9.9817395e-001f, 9.9817300e-001f, 9.9817205e-001f, 9.9817110e-001f,
    9.9817014e-001f, 9.9816919e-001f, 9.9816824e-001f, 9.9816729e-001f,
    9.9816634e-001f, 9.9816539e-001f, 9.9816443e-001f, 9.9816348e-001f,
    9.9816253e-001f, 9.9816158e-001f, 9.9816063e-001f, 9.9815967e-001f,
    9.9815872e-001f, 9.9815777e-001f, 9.9815682e-001f, 9.9815587e-001f,
    9.9815491e-001f, 9.9815396e-001f, 9.9815301e-001f, 9.9815206e-001f,
    9.9815111e-001f, 9.9815015e-001f, 9.9814920e-001f, 9.9814825e-001f,
    9.9814730e-001f, 9.9814635e-001f, 9.9814539e-001f, 9.9814444e-001f,
    9.9814349e-001f, 9.9814254e-001f, 9.9814159e-001f, 9.9814064e-001f,
    9.9813968e-001f, 9.9813873e-001f, 9.9813778e-001f, 9.9813683e-001f,
    9.9813588e-001f, 9.9813492e-001f, 9.9813397e-001f, 9.9813302e-001f,
    9.9813207e-001f, 9.9813112e-001f, 9.9813016e-001f, 9.9812921e-001f,
    9.9812826e-001f, 9.9812731e-001f, 9.9812636e-001f, 9.9812541e-001f,
    9.9812445e-001f, 9.9812350e-001f, 9.9812255e-001f, 9.9812160e-001f,
    9.9812065e-001f, 9.9811969e-001f, 9.9811874e-001f, 9.9811779e-001f,
    9.9811684e-001f, 9.9811589e-001f, 9.9811493e-001f, 9.9811398e-001f,
    9.9811303e-001f, 9.9811208e-001f, 9.9811113e-001f, 9.9811017e-001f,
    9.9810922e-001f, 9.9810827e-001f, 9.9810732e-001f, 9.9810637e-001f,
    9.9810542e-001f, 9.9810446e-001f, 9.9810351e-001f, 9.9810256e-001f,
    9.9810161e-001f, 9.9810066e-001f, 9.9809970e-001f, 9.9809875e-001f,
    9.9809780e-001f, 9.9809685e-001f, 9.9809590e-001f, 9.9809495e-001f,
    9.9809399e-001f, 9.9809304e-001f, 9.9809209e-001f, 9.9809114e-001f,
    9.9809019e-001f, 9.9808923e-001f, 9.9808828e-001f, 9.9808733e-001f,
    9.9808638e-001f, 9.9808543e-001f, 9.9808447e-001f, 9.9808352e-001f,
    9.9808257e-001f, 9.9808162e-001f, 9.9808067e-001f, 9.9807972e-001f,
    9.9807876e-001f, 9.9807781e-001f, 9.9807686e-001f, 9.9807591e-001f,
    9.9807496e-001f, 9.9807400e-001f, 9.9807305e-001f, 9.9807210e-001f,
    9.9807115e-001f, 9.9807020e-001f, 9.9806925e-001f, 9.9806829e-001f,
    9.9806734e-001f, 9.9806639e-001f, 9.9806544e-001f, 9.9806449e-001f,
    9.9806353e-001f, 9.9806258e-001f, 9.9806163e-001f, 9.9806068e-001f,
    9.9805973e-001f, 9.9805878e-001f, 9.9805782e-001f, 9.9805687e-001f,
    9.9805592e-001f, 9.9805497e-001f, 9.9805402e-001f, 9.9805306e-001f,
    9.9805211e-001f, 9.9805116e-001f, 9.9805021e-001f, 9.9804926e-001f

#endif/*_MSC_VER && _M_IX86*/
};

#endif/*EXPTABLE && MULT*/
```

```
/*******************************************************/
/********       Written By Phillip S. Pang      ********/
/*******************************************************/

/*******************************************************/
/******** MD/PhD Candidate, Columbia University ********/
/******** College of Physicians and Surgeons    ********/
/******** Dept. Of Biochemistry and Biophysics  ********/
/*******************************************************/

/*******************************************************/
/********    phillip.pang@stanfordalumni.org    ********/
/*******************************************************/


/*******************************************************/
/********       STATEMENT OF COPYRIGHT          ********/

/*          Copyright 2001 by The Trustees of          */
/*          Columbia University in the City of         */
/*          New York. ALL RIGHTS RESERVED;             */

/*******************************************************/



/*********************************************************************/
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/* THIS CODE WAS TRANSLATED, OPTIMIZED, and WRITTEN by RAUL Shakirov  */
/* from FORTRAN TO C, As a work-for-hire by request of Phillip S. Pang */
/*                                                                   */
/* Contact information for Raul: http://www.imach.uran.ru/rns/       */
/*                                                                   */
/*                                                                   */
/* It has been further modified by Phillip S. Pang                   */
/*                                                                   */
/*********************************************************************/
/*********************************************************************/
/*********************************************************************/


#include <time.h>       /* Header file for standard time functions */

#ifndef CLK_TCK
#define CLK_TCK CLOCKS_PER_SEC
#endif

/* ----------------------------------------------------------------------
    Name:       GetFloatTimer

    Purpose:    Get timer as a second interval between current time
                and given time value.

    Usage:      float timer = GetFloatTimer (0);
                ...
                Some time-consuming process
                ...
                timer = GetFloatTimer (timer);

    Arguments:
      timer   - Time value to be substracted from current time.
---------------------------------------------------------------------- */

float GetFloatTimer (float timer)
{
    /*
        To get correct time difference we need double computation,
        because clock() value may be too great for float presision!
     */

    float time = (float) (((double) clock() / CLK_TCK) - timer);

    /*  Correct time after midnight  */
```

```
    if (time < -1.) time += (3600 * 24);
    /*  Correct impossible precision error. */
    if (time <  0.) time = 0.;
    return time;
}
```

```
/*********************************************************/
/********       Written By Phillip S. Pang       ********/
/*********************************************************/

/*********************************************************/
/******** MD/PhD Candidate, Columbia University ********/
/******** College of Physicians and Surgeons    ********/
/******** Dept. Of Biochemistry and Biophysics  ********/
/*********************************************************/

/*********************************************************/
/********    phillip.pang@stanfordalumni.org    ********/
/*********************************************************/


/*********************************************************/
/********       STATEMENT OF COPYRIGHT          ********/

/*          Copyright 2001 by The Trustees of          */
/*          Columbia University in the City of         */
/*          New York. ALL RIGHTS RESERVED;             */

/*********************************************************/



/*****************************************/
/*****************************************/
/* Estimated Exact Probability Function  **/
/* via Monte Carlo Simulation            **/
/*                                       **/
/* For use when exact (FEXACT function)  **/
/* Fails; and Cochran Conditions are not **/
/* met                                   **/
/*****************************************/
/*****************************************/


#include <math.h>

#include "nrutilp.h"
#include "rcont2p.h"
#include "shevek.h"
#include "definitions.h"



/*************************************************************/
/*************************************************************/
/************   ESTEXACT function       **********************/
/*************************************************************/
/*************************************************************/
/*************************************************************/
/************   ESTEXACT function       **********************/
/*************************************************************/
/*************************************************************/
/*************************************************************/
/************   ESTEXACT function       **********************/
/*************************************************************/
/*************************************************************/



double estExact(int *ROWmatrix, double PRECISION, int numrows, int numcols)
{

int rcont2s(int nrow, int ncol, int *_nrowt,int *_ncolt,int *_matrix);
float rcont2f (void);


int minprecision = 10000;


int *RANDmatrix;
int *rowtot;
int *coltot;
```

```
float *expctd;

double probability;

double temp;
double chisq;
double chisqRAND;
float tprob;                /*VARIABLE IS UNUSED FOR CALCULATIONS*/
int tally;
float ftally;
int numrows2,numcols2,i,j,k,m,n,r, error;
float sum=0.0;

/***********************/
/*allocate memory      */
/***********************/

RANDmatrix=ivector(0,(numrows*numcols -1));
rowtot=ivector(0,(numrows-1));
coltot=ivector(0,(numcols-1));

expctd=vector(0,(numrows*numcols -1));


/***********************/
/*clears memory = 0    */
/***********************/


for (r=0;r < numrows*numcols; r++) {
    *(RANDmatrix + r) = 0;
    *(expctd + r) = 0;
}


/***********************/
/*calc row and col info*/
/***********************/

numrows2=numrows;                           /*Number of rows*/
numcols2=numcols;                           /*and columns.*/

for (i=0;i< numrows;i++) {                  /*Get the row totals.*/
    rowtot[i]=0;                            /*clears memory*/
    for (j=0;j< numcols;j++) {
        rowtot[i] += *(ROWmatrix +(i*numcols + j));
        sum += *(ROWmatrix +(i*numcols + j));
    }
    if (rowtot[i] == 0) --numrows2;             /*Eliminate any zero rows by reducing the
num*/
}


for (j=0;j< numcols;j++) {                  /*Get the column totals.*/
    coltot[j]=0;                            /*clears memory*/
    for (i=0;i< numrows;i++) coltot[j] += *(ROWmatrix +(i*numcols + j));
    if (coltot[j] == 0) --numcols2;             /*Eliminate any zero columns.*/
}


/*********************************************************/
/*** Test of Matrix: must be 2x2 after zero removal *******/
/*********************************************************/

if ((numcols2 < 2) || (numrows2 < 2))   {       /** Chisquare statistical analysis not pos
sible **/
    return MEANINGLESS;
}


/***********************************/
/* Calculate chi of original Rmatrix*/
/***********************************/

chisq=0.0;
```

```
for (i=0;i< numrows;i++) {                  /*Do the chi-square sum.*/
    for (j=0;j< numcols;j++) {
        *(expctd +(i*numcols + j)) = (coltot[j]*rowtot[i])/sum;
        temp= *(ROWmatrix +(i*numcols + j)) - *(expctd +(i*numcols + j));
        chisq += temp*temp  /  (*(expctd +(i*numcols + j)) + TINY);     /*Here TINY guaran
tees that any*/ /*elim. Div. By zero */
    }                               /* eliminated row or column will*/
}                                   /*not contribute to the sum.*/




/*****************************************************/
/* MAIN CYCLE                                        */
/*                                                   */
/* Two parts:                                        */

/*First, min. precision of 1/minprecision            */
/*Then, count till PRECISION, unless tally >= 10     */

/*****************************************************/


/** added to jump over scoring -- see shevek.c file and openfile function **/

if (PRECISION == 42)    {   /*occurs when prec == 42 */
    minprecision = 10;
}

/*************************************************************************/



tally = 0;
error = 0;
error = rcont2s(numrows,numcols,rowtot,coltot,RANDmatrix);


if (error == 0) {

for (k=1; k<=minprecision; k++) {


    /*********************************************************/
    /*** generate random matrix using marginal frequencies ***/
    /*********************************************************/

    tprob = rcont2f();

    /*************************************************/
    /*** determine chisquare for that random matrix ***/
    /*************************************************/

    chisqRAND = 0.0;

    for (m=0;m< numrows;m++) {                                          /*Do the c
hi-square sum.*/
        for (n=0;n< numcols;n++) {
            temp= *(RANDmatrix +(m*numcols + n)) - *(expctd +(m*numcols + n));
            chisqRAND += temp*temp  /  (*(expctd +(m*numcols + n)) + TINY);      /*Here
 TINY guarantees that any*/
                                                                        /*elim. Di
v. By zero */
        }                                                               /* elimina
ted row or column will*/
    }                                                                   /*not cont
ribute to the sum.*/


    /*************************************************/
    /* COMPARE original matrix to random matrix      */
    /*************************************************/
```

```
    if (chisqRAND >= chisq)
        tally += 1;

}


while ((tally<10) && (k<=PRECISION))    {

    /*********************************************************/
    /*** generate random matrix using marginal frequencies ***/
    /*********************************************************/

    tprob = rcont2f();

    /*************************************************/
    /*** determine chisquare for that random matrix ***/
    /*************************************************/

    chisqRAND = 0.0;

    for (m=0;m< numrows;m++) {                                          /*Do the c
hi-square sum.*/
        for (n=0;n< numcols;n++) {
            temp= *(RANDmatrix +(m*numcols + n)) - *(expctd +(m*numcols + n));
            chisqRAND += temp*temp  /  (*(expctd +(m*numcols + n)) + TINY);         /*Here
 TINY guarantees that any*/
                                                                        /*elim. Di
v. By zero */
        }                                                               /* elimina
ted row or column will*/
    }                                                                   /*not cont
ribute to the sum.*/


    /*************************************************/
    /* COMPARE original matrix to random matrix       */
    /*************************************************/


    if (chisqRAND >= chisq)
        tally += 1;

    k = k+1;

}


}

else return MEANINGLESS;



/******************/
/*END OF MAIN CYCLE*/
/******************/


if (tally != 0) {
    ftally = (float) tally;
    probability = ftally/ (k-1);
}
else
    probability = 1.0/PRECISION;


free_ivector(coltot,0,(numcols-1));
free_ivector(rowtot,0,(numrows-1));
free_ivector(RANDmatrix,0,(numrows*numcols -1));
free_vector(expctd,0, (numrows*numcols -1));
```

```
return probability;

}
```

```
/*********************************************************/
/********       Written By Phillip S. Pang      ********/
/*********************************************************/

/*********************************************************/
/******** MD/PhD Candidate, Columbia University ********/
/******** College of Physicians and Surgeons    ********/
/******** Dept. Of Biochemistry and Biophysics  ********/
/*********************************************************/

/*********************************************************/
/********    phillip.pang@stanfordalumni.org    ********/
/*********************************************************/


/*********************************************************/
/********       STATEMENT OF COPYRIGHT          ********/

/*          Copyright 2001 by The Trustees of          */
/*          Columbia University in the City of         */
/*          New York. ALL RIGHTS RESERVED;             */

/*********************************************************/



/************************************************************************/
/************************************************************************/
/************************************************************************/
/*                                                                      */
/* THIS CODE WAS TRANSLATED, OPTIMIZED, and WRITTEN by RAUL Shakirov    */
/* from FORTRAN TO C, As a work-for-hire by request of Phillip S. Pang  */
/*                                                                      */
/* Contact information for Raul: http://www.imach.uran.ru/rns/          */
/*                                                                      */
/*                                                                      */
/* It has been further modified by Phillip S. Pang                      */
/*                                                                      */
/************************************************************************/
/************************************************************************/
/************************************************************************/



#include <stdio.h>      /* Header file for standard io functions */
#include <stdlib.h>     /* Header file for standard functions */
#include "rcont2p.h"     /* Header file for rcont2 functions */

/* ---------------------------------------------------------------------
    Name:       prcerr

    Purpose:    Print an error message to console and exit.

    Usage:      prcerr (icode, mes)

    Arguments:
      icode   - Int code for the error message.                 (Input)
      mes     - Character string containing the error message.  (Input)
--------------------------------------------------------------------- */

void prcerr (int icode, const char *mes)
{
    fprintf (stderr, "\nRCONT2 ERROR: %d %s\n", icode, mes);
                              /* Print diagnostic to console */
    exit (2);                 /* Close all files and exit with code 1 */
}

/* ---------------------------------------------------------------------
    Name:       prcerrN

    Purpose:    Print error message N to console and exit.

    Usage:      prcerrN()
--------------------------------------------------------------------- */
```

```
void prcerr1 (void)
{
    prcerr (1, "Number of rows must be greater than 1.");
}

void prcerr2 (void)
{
    prcerr (2, "Number of columns must be greater than 1.");
}

void prcerr3 (void)
{
    prcerr (3, "Row totals must be greater than 0.");
}

void prcerr4 (void)
{
    prcerr (4, "Column totals must be greater than 0.");
}

void prcerr5 (void)
{
    prcerr (5, "Sample size exceeds 5000.");
}

void prcerr6 (void)
{
    prcerr (6, "Sum of row and column totals must be the same.");
}
```

```
/*******************************************************/
/********       Written By Phillip S. Pang      ********/
/*******************************************************/

/*******************************************************/
/******** MD/PhD Candidate, Columbia University ********/
/******** College of Physicians and Surgeons    ********/
/******** Dept. Of Biochemistry and Biophysics  ********/
/*******************************************************/

/*******************************************************/
/********    phillip.pang@stanfordalumni.org    ********/
/*******************************************************/


/*******************************************************/
/********       STATEMENT OF COPYRIGHT          ********/

/*          Copyright 2001 by The Trustees of          */
/*          Columbia University in the City of         */
/*          New York. ALL RIGHTS RESERVED;             */

/*******************************************************/


/**************************************************************************/
/**************************************************************************/
/**************************************************************************/
/*                                                                        */
/*      Portable Random Number Generators                                 */
/*                                                                        */
/*      Gerald P. Dwyer Jr. and K.B. Williams                             */
/*      Federal Reserve Bank of Atlanta                                   */
/*      Working Paper 99-14                                               */
/*      October 1999                                                      */
/*                                                                        */
/*          NOTE: PORTIONS OF THIS CODE IN THIS FILE have been taken      */
/*          from the above paper                                          */
/*          the above paper.                                              */
/*                                                                        */
/*                                                                        */
/*                                                                        */
/**************************************************************************/

/* CONTACT THE ABOVE AUTHORS AT:

Gerald P. Dwyer Jr.,
Research Department,
Federal Reserve Bank of Atlanta,
104 Marietta Street, NW,
Atlanta, Georgia 30303-2713,
404/614-7095,
404/521-8810 (fax),
gerald.p.dwyer@atl.frb.org
(or dwyerg@clemson.edu);

K.B. Williams,
412 Magnolia Avenue,
Melbourne Beach, Florida 32951-2000
kbwms@aol.com.

*/

/**************************************************************************/
/**************************************************************************/


/* RandComb.cpp
combination multiplicative random number generator
takes difference between two random numbers
rand1 and rand2 can be initialized outside of file */


static const long mod1 = 2147483647 ;
static const long mult1 = 65670 ;
static const long q1 = 2147483647 / 65670;  /* mod1/ mult1 ;*/
```

```
static const long r1 = 2147483647-65670 * (2147483647 / 65670);  /*mod1 - mult1 * q1 ;*/


static const long mod2 = 2147483587 ;
static const long mult2 = 44095 ;
static const long q2 = 2147483587/44095; /* mod2 / mult2 ; */
static const long r2 = 2147483587 - 44095 *  (2147483647 / 65670); /*mod2 - mult2 * q1 ;*/


long rand1=1, rand2=1 ;



/*************************************************************/
/************   PANGRAND function                 **********/
/*************************************************************/
long double pangrand(void)
{
    long double randtemp1;

    /*long double randtemp2;*/

    long RandComb(void);

    randtemp1 =   ((long double) RandComb())  /  ( (long double) (mod1-1) );

    /**randtemp2 = randtemp1 /  ( (long double) (mod2-1) );  **/

    return randtemp1;  /*  2*randtemp2;  */

}

/*************************************************************/
/************   GENRRAND function                 **********/
/*************************************************************/


/* generate the next value in sequence from generator using approximate factoring */

long GenrRand(long rand, long modulus, long mult, long q, long r)
{

    long temp ;

    temp = rand / q ;
    temp = mult * (rand - temp * q) - temp * r ;

    if (temp < 0)   temp += modulus ;

    return temp ;
}



/*************************************************************/
/************   RANDCOMB function                 **********/
/*************************************************************/

/* get a random number */

long RandComb(void)
{
    long RandNum ;
    long GenrRand(long rand, long modulus, long mult, long q, long r);


    rand1 = GenrRand(rand1, mod1, mult1, q1, r1) ;
    rand2 = GenrRand(rand2, mod2, mult2, q2, r2) ;
    RandNum = rand1 - rand2 ;

    if (RandNum < 0)    RandNum += mod1-1 ;

    return RandNum ;
}
```

```
/*****************************************************/
/********       Written By Phillip S. Pang      ********/
/*****************************************************/

/*****************************************************/
/******** MD/PhD Candidate, Columbia University ********/
/******** College of Physicians and Surgeons    ********/
/******** Dept. Of Biochemistry and Biophysics  ********/
/*****************************************************/

/*****************************************************/
/********    phillip.pang@stanfordalumni.org    ********/
/*****************************************************/


/*****************************************************/
/********       STATEMENT OF COPYRIGHT          ********/

/*          Copyright 2001 by The Trustees of          */
/*          Columbia University in the City of         */
/*          New York. ALL RIGHTS RESERVED;             */

/*****************************************************/



/*********************************************************************/
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/* THIS CODE WAS TRANSLATED, OPTIMIZED, and WRITTEN by RAUL Shakirov  */
/* from FORTRAN TO C, As a work-for-hire by request of Phillip S. Pang */
/*                                                                   */
/* Contact information for Raul: http://www.imach.uran.ru/rns/       */
/*                                                                   */
/*                                                                   */
/* It has been further modified by Phillip S. Pang                   */
/*                                                                   */
/*********************************************************************/
/*********************************************************************/
/*********************************************************************/



/*****************************************************************/
/*****************************************************************/
/*****************************************************************/
/*                                                               */
/*  The algorithms found within this file may be derivatives     */
/*  of source code obtained from:                                */
/*    ALGORITHM AS 159  APPL. STATIST. (1981) VOL.30, NO.1       */
/*  See: Patefield, Applied Statistics 30:91-97 (1981)           */
/*                                                               */
/*****************************************************************/
/*****************************************************************/
/*****************************************************************/



#include <float.h>      /* Header file for standard math functions */
#include <math.h>       /* Header file for standard math functions */
#include "rcont2p.h"     /* Header file for rcont2 functions */

/* ------------------------------------------------------------------------
    ALGORITHM AS 159  APPL. STATIST. (1981) VOL.30, NO.1

    Name:       rcont2s
                rcont2f

    Purpose:    Generate random two-way table with given marginal totals

    Usage:      srand ((unsigned) time (NULL))
                rcont2s (nrow, ncol, nrowt, ncolt, matirix)
                rcont2f()
```

```
       nrow   - The number of rows in the contingecy table.       (Input)
       ncol   - The number of columns in the contingency table.   (Input)
       nrowt  - Vector of length nrow contining row totals
                of the observed table                             (Input)
       ncolt  - Vector of length ncol contining column totals
                of the observed table                             (Input)
       matrix - nrow by ncol matrix for the contingency table
                to be generated                                  (Output)

    Notes:      srand   Standard C function which is used to set
                        pseudo-random seed (requires stdlib.h).

                time    Standard C function which is used to get
                        current time (requires time.h).

                rcont2s stores arguments and returns error code


                rcont2f does same as rcont2 but uses fast
                        table-driven exp function with 20 bit
                        presizion instead of standard exp function.
---------------------------------------------------------------------- */



/************************************************************************/
/************************************************************************/

/*
    Static variables and arrays are kept between calls of
    function because they are allocated and inilialized once
    when the program is loaded. Default initial value - 0.
*/

/*  Static variables for arguments. */

static int nrowm;               /* The number of rows - 1 */
static int ncolm;               /* The number of cols - 1 */
static int *nrowt;        /* Vector of row totals */
static int *ncolt;        /* Vector of col totals */
static int *matrix;       /* Matrix for the contingency table */
static int *jwork;        /* Last row of matrix */
static int ntotal;              /* Total sum */

/*
    Static work array for optimization purposes
    log - log-factorials (log(1) + log(2) + ... log(n))
    div - dividors       (1/n)
*/

static int nfill = 0;           /* Index of filled part of fact */
                                /* Work array */
static struct {float log, div; } fact [maxtot + 1];


/************************************************************************/
/************************************************************************/




/************************************************************************/
/* ---------------------------------------------------------------------
    Name:       rcont2s

    Purpose:    Prepare to construct random matrix.

    Usage:      Ref. top of file.

    Returns:    Error code:
                   1 Number of rows must be greater than 1.
                   2 Number of columns must be greater than 1.
                   3 Row totals must be greater than 0.
```

```
                    4 Column totals must be greater than 0.
                    5 Sample size exceeds 5000.
                    6 Sum of row and column totals must be the same

    External referencies:
       log, prcerr1..6
------------------------------------------------------------------------------ */
/*****************************************************************************/



int rcont2s (int nrow, int ncol,
             int *_nrowt,  int *_ncolt,
             int *_matrix)
{
    /* Local variables */

    int ntot = 0;

    /* Check for faults and prepare for future calls of rcont2x */

    if ((nrowm = nrow - 1) <= 0)
    {
        return 1;                          /* Error */
    }

    if ((ncolm = ncol - 1) <= 0)
    {
        return 2;                          /* Error */
    }

    /* Store arrays. */

    nrowt  = _nrowt;
    ncolt  = _ncolt;
    matrix = _matrix;

    /* EXPTRTO macro calculates debug pointer to sub-array. */

    jwork  = EXPTRTO (int, matrix + nrowm * ncol, ncol);

    /* Compute total sum */

    {
        int i, ilim;                  /* Loop index and limit */
        ilim = nrow;
        for (i = 0; i < ilim; ++i)
        {
            int n = nrowt [i];
            if (n <= 0)
            {
                return 3;                  /* Error */
            }
            ntot += n;
        }
    }

    if (ntot > maxtot)
    {
       return 5;                           /* Error */
    }

    /* Calculate log-factorials and dividors */

    {
        /* Local variables for this blick */

        int n = nfill;                     /* Loop index */
        float f = fact [n].log;            /* Fact value */

        for (++n; n <= ntot; ++n)
        {
            f += (float) log ((float) n);
            fact [n].log = f;
            fact [n].div = (float) 1./((float) n);
```

```
        }

        nfill = ntot;
    }

    /* Store total sum */

    ntotal = ntot;

    /* Check if row and column sums are the same */

    {
        int j, jlim;                /* Loop index and limit */
        jlim = ncol;
        for (j = 0; j < jlim; ++j)
        {
            int n = ncolt [j];
            if (n <= 0)
            {
                return 4;           /* Error */
            }
            ntot -= n;
        }
    }

    if (ntot != 0)
    {
       return 6;                    /* Error */
    }

    return 0;                       /* No errors */
}



/*****************************************************************************/
/* ---------------------------------------------------------------------------
    Name:       rcont2f

    Purpose:    Construct random matrix using table-driven
                20 bit exp function.

    Usage:      Ref. top of file.

    Returns:    Probability of observing the sample table
                as float value.

    External referencies:
       rand, fastexp2
--------------------------------------------------------------------------- */
/*****************************************************************************/


float rcont2f (void)

#define FASTEXP(value) fastexp2(value)  /* Use table-driven exp function */


{
    /* Local variables */

    int i, j;                   /* Loop index and limit */
    EXPTR(int) matp = matrix;   /* Current element to be calculated */

    float hop = 1.;             /* Probabilty value */

    int jc = ntotal;            /* Total sum for subsequent rows */
    int ib;                     /* Remainder of total sum */

    /* Copy ncolt to last row of matrix (jwork) */

    for (j = 0; j < ncolm; ++j)
    {
       jwork [j] = ncolt [j];
```

```
}

/* Cycle for rows */

for (i = 0; i < nrowm; ++i)
{
    /* Local variables for this cycle */

    int ic = jc;            /* Remainder of total sum */
    int ia = nrowt [i];     /* Remainder of row total */

    /* Update jc */

    jc -= ia;

    /* Cycle for elements of the row */

    for (j = 0; j < ncolm; ++j)
    {
        /*  Local variable for this cycle */

        int nlm;            /* Value of element to be computed */

        /* Test for zero entries in matrix */

        if (ic != 0)
        {
            /* Generate pseudo-random number */

            long double dummy;

            /* Update remainders */

            const int ie = ic;          /* Remainder of total sum */
            const int id = jwork [j];   /* Remainder of column total */
            const int ii = (ib = ic - ia) - id;

            /* Calulate initial value of nlm */

            float fnlm = (float) ia * (float) id * fact [ie].div
                    #if defined(_MSC_VER) && defined(_M_IX86)
                       * (float) (1. + FLT_EPSILON);
                    #else
                       + (float) 0.5;
                    #endif

            /* Update ic */

            ic -= id;

            dummy = (long double) RAND;

            /* Compute conditional expected value of matrix [i] [j] */

            for (;;)
            {
                /* Local variables for this cycle */

                float x, y, sumprb;       /* Work values */
                int nll;                  /* nlm, nlm-1, ... */

            #if defined(_MSC_VER) && defined(_M_IX86)
                __asm fld   fnlm
                __asm fistp nlm
            #else
                nlm = (int) fnlm;
            #endif

                x = (float) FASTEXP (
                        fact [ia].log + fact [ib].log +
                        fact [ic].log + fact [id].log - fact [ie].log -
                        fact [id - nlm].log - fact [ia - nlm].log -
                        fact [ii + nlm].log - fact [nlm].log
                        );

                if (x >= dummy)
```

```
{
    hop *= x;
    break /* computing for */;
}

y = x;
sumprb = x;
nll = nlm;

for (;;)
{
    /* Local variables for this cycle */

    int lsp, lsm;           /* Work values */

    if ((lsp = id - nlm) != 0 &&
        (lsm = ia - nlm) != 0)
    {
        /* Increment entry in row i, column j */

        ++nlm;

        x *= ((float) lsp * fact [nlm].div) *
             ((float) lsm * fact [ii + nlm].div);

        if ((sumprb += x) >= dummy)
        {
            hop *= x;
            goto MATP /* break computing for */;
        }
    }
    else
    {
        while ((lsp = nll) != 0 &&
               (lsm = ii + nll) != 0)
        {
            /* Decrement entry in row i, column j */

            --nll;

            y *= ((float) lsp * fact [id - nll].div) *
                 ((float) lsm * fact [ia - nll].div);

            if ((sumprb += y) >= dummy)
            {
                hop *= y;
                nlm = nll;
                goto MATP /* break computing for */;
            }
        }

        break /* for */;
    }

    if ((lsp = nll) != 0 &&
        (lsm = ii + nll) != 0)
    {
        /* Decrement entry in row i, column j */

        --nll;

        y *= ((float) lsp * fact [id - nll].div) *
             ((float) lsm * fact [ia - nll].div);

        if ((sumprb += y) >= dummy)
        {
            hop *= y;
            nlm = nll;
            goto MATP /* break computing for */;
        }
    }
    else
    {
        while ((lsp = id - nlm) != 0 &&
               (lsm = ia - nlm) != 0)
        {
```

```
                                /* Increment entry in row i, column j */

                                ++nlm;

                                x *= ((float) lsp * fact [nlm].div) *
                                     ((float) lsm * fact [ii + nlm].div);

                                if ((sumprb += x) >= dummy)
                                {
                                    hop *= x;
                                    goto MATP /* break computing for */;
                                }
                            }
                            break /* for */;
                        }
                    }

                    /* Generate new pseudo-random number */

                    dummy = (long double) sumprb*RAND;
                }

            MATP:
                /* Set element of matrix */

                *matp++ = nlm;
                jwork [j] -= nlm;
                ia -= nlm;
            }
            else /* ic == 0 */
            {
                /* All remaining elements in the row are to be zero */

                for (; j < ncolm; ++j)
                {
                    *matp++ = 0;
                }

                /*
                   Probably, it is sufficient to set ib = 0,
                   because ia should be 0
                */

                ib = ic - ia;
                ia = 0;

                break /* for */;
            }
        }

        /* Set last element in the row */

        *matp++ = ia;
    }

    /* Compute last element in last row of matrix */

    jwork [j] = ib - jwork [j - 1];

    return hop;
}


#undef  FASTEXP                         /* Undef fastexp */
```

```
/*********************************************************/
/********        Written By Phillip S. Pang       ********/
/*********************************************************/

/*********************************************************/
/******** MD/PhD Candidate, Columbia University ********/
/******** College of Physicians and Surgeons    ********/
/******** Dept. Of Biochemistry and Biophysics  ********/
/*********************************************************/

/*********************************************************/
/********    phillip.pang@stanfordalumni.org    ********/
/*********************************************************/


/*********************************************************/
/********        STATEMENT OF COPYRIGHT          ********/

/*          Copyright 2001 by The Trustees of          */
/*          Columbia University in the City of         */
/*          New York. ALL RIGHTS RESERVED;             */

/*********************************************************/



/************************************************************************/
/************************************************************************/
/************************************************************************/
/*                                                                      */
/* THIS CODE WAS TRANSLATED, OPTIMIZED, and WRITTEN by RAUL Shakirov    */
/* from FORTRAN TO C, As a work-for-hire by request of Phillip S. Pang  */
/*                                                                      */
/* Contact information for Raul: http://www.imach.uran.ru/rns/          */
/*                                                                      */
/*                                                                      */
/* It has been further modified by Phillip S. Pang                      */
/*                                                                      */
/************************************************************************/
/************************************************************************/
/************************************************************************/



#include <malloc.h>     /* Header file for standard memory functions */
#include <stdio.h>      /* Header file for standard io functions */
#include <stdlib.h>     /* Header file for standard functions */
#include "support.h"    /* Header file for support functions */

/* ---------------------------------------------------------------------
   Static variable to check if memory heap is properly deallocated.
--------------------------------------------------------------------- */

static int syscount;

/* ---------------------------------------------------------------------
    Name:       sysalloc

    Purpose:    Allocate memory for array.

    Usage:      sysalloc (nitems, size)

    Arguments:
       nitems - Number of intes in array
       size   - size of item

    Result:    - Pointer to allocated array

    External referencies:
       syserr40
--------------------------------------------------------------------- */

void* sysalloc (int nitems, int size)
{
   void *p = calloc (nitems, size);
```

```
   if (p == NULL) syserr40();

   ++syscount;

   return p;
}

/* ---------------------------------------------------------------------
    Name:       sysfree

    Purpose:    Deallocate memory, allocated by sysalloc,

    Usage:      sysfree (p)

    Arguments:
      p      - Pointer to deallocated array
--------------------------------------------------------------------- */

void sysfree (void *p)
{
   if (p != NULL)
   {
      --syscount;

      free (p);
   }
}

/* ---------------------------------------------------------------------
    Name:       syschk

    Purpose:    Check if memory is properly deallocated.

    Usage:      syschk()
--------------------------------------------------------------------- */

void syschk (void)
{
   if (syscount)                        /* Print diagnostic to console */
   {
      fprintf (stderr, "\nSYSTEM WARNING: Memory leak detected.\n"
                       "It's an internal error. Ask developers.\n");
   }
}

/* ---------------------------------------------------------------------
    Name:       syserr

    Purpose:    Print an error message to console and exit.

    Usage:      prterr (icode, mes)

    Arguments:
      icode   - Int code for the error message.                (Input)
      mes     - Character string containing the error message. (Input)
--------------------------------------------------------------------- */

void syserr (int icode, const char *mes)
{
    fprintf (stderr, "\nSYSTEM ERROR: %d %s\n", icode, mes);
                              /* Print diagnostic to console */
    exit (2);                 /* Close all files and exit with code 1 */
}

/* ---------------------------------------------------------------------
    Name:       syserrN

    Purpose:    Print error message N to console and exit.

    Usage:      syserrN()
--------------------------------------------------------------------- */

void syserr40 (void)
{
    syserr (40, "Out of workspace.\n"
                "Select system with more RAM available");
```

```
}

void syserr41 (void)
{
    syserr (41, "Automatic expansion of array is not permissible.\n"
                "It's an internal error. Ask developers.");
}

void syserr42 (void)
{
    syserr (42, "Index is out of range.\n"
                "It's an internal error. Ask developers.");
}

void syserr43 (void)
{
    syserr (43, "Null pointer indirection.\n"
                "It's an internal error. Ask developers.");
}

void syserr50 (void)
{
    syserr (50, "Integer overflow detected.\n"
                "It's an internal error. Ask developers.");
}

/* ---------------------------------------------------------------------------
    Internal trace counters
--------------------------------------------------------------------------- */

int pcount [PCOUNT];

/* ---------------------------------------------------------------------------
    Name:       pcinit

    Purpose:    Initilaize values of counters.

    Usage:      pcinit()
--------------------------------------------------------------------------- */

void pcinit (void)
{
    int i;

    /* Initialize counters */

    for (i = 0; i < PCOUNT; i++)
    {
        pcount [i] = 0;
    }
}

/* ---------------------------------------------------------------------------
    Name:       ptrace

    Purpose:    Increase trace counter.

    Usage:      ptrace (n)

    Returns:    Counter value.

    Arguments:
       n      - Counter
--------------------------------------------------------------------------- */

#ifndef ptrace  /* Might be implemented as a macro in fexact.h */

int ptrace (int n)
{
    /* Increase counter */

    return (++pcount [n]);
}

#endif
```

```
/* ---------------------------------------------------------------------
    Name:       pctype

    Purpose:    Type profiling counters to console.

    Usage:      pctype()
--------------------------------------------------------------------- */

void pctype (void)
{
    int i;

    /* Type carriage return */

    fprintf (stderr, "\r");

    /* Type counters */

    for (i = 0; i < PCOUNT; i++)
    {
        if (pcount [i]) fprintf (stderr, "%d:%d ", i, pcount [i]);
    }
}
```

```
/*********************************************************/
/********       Written By Phillip S. Pang       ********/
/*********************************************************/

/*********************************************************/
/******** MD/PhD Candidate, Columbia University ********/
/******** College of Physicians and Surgeons    ********/
/******** Dept. Of Biochemistry and Biophysics  ********/
/*********************************************************/

/*********************************************************/
/********    phillip.pang@stanfordalumni.org    ********/
/*********************************************************/


/*********************************************************/
/********       STATEMENT OF COPYRIGHT          ********/

/*          Copyright 2001 by The Trustees of          */
/*          Columbia University in the City of         */
/*          New York. ALL RIGHTS RESERVED;             */

/*********************************************************/



#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

#include "nrutilp.h"
#include "shevek.h"
#include "definitions.h"


double POFF;
double CRV_CUTOFF;

int FILEL1;
int FILEL2;

/*************************************************************/
/************   APPLY_THRESHOLDS function   ******************/
/*************************************************************/


void apply_thresholds(int filelength)
{

    double  *openfile2(int pass,int filelength);
    int prelimscan(double *input, int filelength);

    /* void eliminator(double *input,int filelength);*/

    void  eliminator2(double *input,int filelength);

    double  *input;
    double *input2;
    int filelength2;
    void    message(int number);


    input = openfile2(0,filelength);                          /*OPENS THEN READS FILE*/
                                                             /*returns pointer to sequence
data*/

    filelength2 = prelimscan(input,FILEL1);                  /*Outputs File PVscan.txt that is
preliminary list of interactions*/


    message(4);              /*PRINT TO SCREEN ELIMINATE INTERSECTING INTERACTIONS*/


    input2 = openfile2(1,filelength2);                       /*Opens file just created -- PVsca
n.txt*/
```

```
    /* eliminator(input2,filelength2);              */      /*eliminates intersecting inte
ractions*/

    eliminator2(input2,FILEL2);

    free_dvector(input,0,FILEL1*COLS);  /*frees memory allocated for input file */

    free_dvector(input2,0,FILEL2*COLS);

}


/************************************************************/
/************   OPENFILE2 function   ************************/
/************************************************************/

double  *openfile2(int pass, int filelength)
{

    double  *read_input2(FILE *ifile, int pass);
    double  *input;
    char    filename[50];

    extern double POFF;
    extern double CRV_CUTOFF;

    FILE * inputfile = NULL;

    int found_file = 0;

    while (found_file == 0) {                       /*Queries for input file*/

        if (pass == 0)  {

        printf("\nPlease enter datafile for threshold application (alldataXXX.txt):");

        }

        else    printf("\nENTER preliminary predictions file name (PVscan.txt):");



        scanf("%s", filename);
        inputfile = fopen(filename, "r");
        if (inputfile == NULL)   {
            printf("\nFAILURE TO OPEN OR LOCATE FILE!!!\n");
            found_file = 0;
        }
        else
            found_file = 1;
    }


    if (pass == 0)  {

        printf("\nTHRESHOLD ENTRY (use numbers above as defaults):");
        printf("\nPlease enter a -log(P) threshold (range:0-inf):");
        scanf("%lf",&POFF);                             /*query for p-value parameter*/
                                                        /*NOTE: not currently used */

        printf("\nPlease enter a V threshold (range:0-1):");
        scanf("%lf",&CRV_CUTOFF);                       /*query for Cramer's V paramerter*
/
                                                        /*NOTE: not currently used */


    }


    input = read_input2(inputfile,pass);
    fclose(inputfile);

    return input;

}
```

```
/*************************************************************/
/************   READ_INPUT2 function   ***********************/
/*************************************************************/


double  *read_input2(FILE *ifile, int pass)
{
    void exitprogram();

    double *dvector(long nl, long nh);
    double  *input;

    double inputtemp;

    int i;

    int x;

    int filelength =0;

/*** Memory allocation: determining file size ***/

    printf("\nAllocating Memory for data file.\n");

    x = fgetc(ifile);
    while (x != EOF) {

        if (x == '\n')  filelength += 1;
        x = fgetc(ifile);
    }

    if (filelength == 0)    {
        printf("\nFILE IS EMPTY: ERROR: CONTACT PROGRAMER: phillip.pang@stanfordalumni.org
\n");
        exitprogram();
        exit(11);
    }


    if (pass == 0)  {   /* alldata file */
        filelength = filelength - 3;  /** see chi_analysis for output file extras of allda
ta.txt **/

        FILEL1 = filelength;
    }

    if (pass == 1) {    /*PVscan file */
        filelength = filelength;  /*see output from threshold.c*/

        FILEL2 = filelength;
    }



    input = dvector(0,filelength*COLS);


    if (input == NULL) {
        printf("\nDynamic Memory Allocation FAILURE!!\n");
        exitprogram();
        exit(1);
    }
    else
        printf("Allocation Completed.\n");



    /*** INPUT FILE INTO MEMORY ***/

    fseek(ifile, 0, SEEK_SET);  /*start reading from begining of file*/
    printf("Attempting to Read File. Please Wait.\n");

    i = 0; /*start at begining of memory*/
```

```
    input[0] = 0;
    inputtemp = 0;



    while (i < filelength*COLS) {

        fscanf(ifile, "%lf", &inputtemp);
        input[i] = inputtemp;

        i = i+1;
    }


    printf("\n\nFile Read.\n");

    return input;
}



/*************************************************************/
/************   PRELIMSCAN function     **********************/
/*************************************************************/


int prelimscan(double *input, int filelength)
{


extern double POFF;
extern double CRV_CUTOFF;
int filelength2 = 0;



char    outfile[30] = "PVscan";     /*output file for data*/
char    fullname[40];                       /*output file for data*/


FILE * output = NULL;                       /*output file initiation*/


/*i = row and col = col*/
int i=0;


int check = -1;
sprintf(fullname, "%s.txt", outfile);


output = fopen(fullname, "w");

if (output == NULL)      {
    printf("\nFailure to Create PVscan File.\n");
    exit(2);
}


for (i = 0; i < filelength; i++)    {

    if ( ( (*(input + (i * COLS + col1) ) ) >= CRV_CUTOFF)  &&  ( (*(input + (i * COLS + c
ol2))) >= POFF)  ) {

            filelength2 = filelength2 + 1;

            fprintf(output, "%d\t%d\t%.3f\t%.5f\t%.20lf\t%.5f\t%f\t%.5f\n",
                (int) (*(input + (i * COLS + 0))),
                (int) (*(input + (i * COLS + 1))),
                (float) (*(input + (i * COLS + 2))),
                (float) (*(input + (i * COLS + 3))),
                (*(input + (i * COLS + 4))),
                (float) (*(input + (i * COLS + 5))),
                (float) (*(input + (i * COLS + 6))),
                (float) (*(input + (i * COLS + 7)))  );
```

```
    }

}

fclose(output);

return filelength2;

}
```

```
/*******************************************************/
/********       Written By Phillip S. Pang      ********/
/*******************************************************/

/*******************************************************/
/******** MD/PhD Candidate, Columbia University ********/
/******** College of Physicians and Surgeons    ********/
/******** Dept. Of Biochemistry and Biophysics  ********/
/*******************************************************/

/*******************************************************/
/********    phillip.pang@stanfordalumni.org    ********/
/*******************************************************/


/*******************************************************/
/********       STATEMENT OF COPYRIGHT          ********/

/*          Copyright 2001 by The Trustees of          */
/*          Columbia University in the City of         */
/*          New York. ALL RIGHTS RESERVED;             */

/*******************************************************/
```



```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <math.h>

#include "nrutilp.h"
#include "shevek.h"
#include "definitions.h"


void    eliminator2(double *input,int filelength)
{

    void pathfinder(int fileposition, int group, double *datafile, int filemax);
    void    message(int number);

    int group = -1;

    float dferror;
    float dftemp;

    int groupticker;
    int groupnumber;

    int i;

    double crambest;
    int firstv;

    int firstp;
    double lprobbest;

    int firstdf;
    int dfbest;
    int best;

    int printcounter = 0;

    char    outfile[30] = "Predict";        /*output file for data*/
    char    fullname[40];                       /*output file for data*/

    FILE * output = NULL;                       /*output file initiation*/
```

```
    /* Part One -- FIND ALL NETWORKS of INTERACTRIONS/REDUNDANT INTERACTIONS*/
    /*for every postion not found already as part of a network, find its network, and assi
gn own group name */


    for (i=0;i<filelength;i++) {

        if (  (*(input + (i * COLS + col4))) >= 0    )  {   /*not assigned to a group*/

            pathfinder(i, group, input, filelength);                /*find its network, an
d assign a group name to entire network*/

            group = group - 1;                                          /*get new group na
me*/

        }
    }




    /* Part Two -- Having assigned a group to every preliminary association interaction, f
ind bestin each group */
    /* best V to error of .05; then best P; then lowest DF */


    group = group + 1;      /*since last assignment will decrement */

    groupticker = group;


    while (groupticker < 0) {           /*for each and every group, find best V */

        firstp = NO;
        groupnumber = 0;


/*find best P*/
/*NOTE for P, BEST is HIGHEST*/


        for (i=0;i<filelength;i++) {


            if ( (*(input + ( i * COLS + col4))) == groupticker ) {

                groupnumber = groupnumber + 1;

            }


            if ( firstp == NO && (*(input + ( i * COLS + col4))) == groupticker  )  {   /*
find first member of this group */

                lprobbest = (*(input + (i * COLS + col2)));     /*assign best P to first P
 in group*/
                firstp = YES;

            }

            else if ( firstp == YES && (*(input + ( i * COLS + col4))) == groupticker  )
  {   /*find other members of this group */

                if ( lprobbest  < (*(input + ( i * COLS + col2)))     )     {       /*comp
are P's within group*/
                    lprobbest = (*(input + ( i * COLS + col2)));        /*if new one i
s higher, reasign best */
                }
            }

        }
```

```
/*eliminate all but best P, within Perror*/

        for (i=0;i<filelength;i++) {

            if (     (*(input + ( i * COLS + col4))) == groupticker
  /*same group*/
                &&   (*(input + ( i * COLS + col2)))  <  (lprobbest - PERROR)   )   {
  /*less then best + error*/

                *(input + (i * COLS + col1)) = NEGONE;
                groupnumber = groupnumber - 1;

            }
        }

/*is there more than one best left?*/
/*then judge by df */
/*NOTE for df, BEST is lowest */


        if (groupnumber > 1) {          /*i.e. more than one member in the group left -- n
eed to use logP as judge */

            firstdf = NO;

/*find best DF*/

            for (i=0;i<filelength;i++) {

                if (     (*(input + ( i * COLS + col4))) == groupticker
  /*find first member of group and sub group*/
                    &&   (*(input + ( i * COLS + col1)))  != NEGONE
  /*sub-group of best V, within error*/
                    &&    firstdf == NO                                  ) {



                    dfbest = (int) (*(input + (i * COLS + col3)));      /*assign best df t
o first df in group and sub group*/
                    firstdf = YES;

                }


                else if (       firstdf == YES                                          /*find
 other members of this group */
                            &&   (*(input + ( i * COLS + col4))) == groupticker
                            &&   (*(input + ( i * COLS + col1)))  != NEGONE  ) {


                        if ( dfbest  > ((int) (*(input + ( i * COLS + col3))))     )
  {       /*compare DF's within group*/
                            dfbest = (int) (*(input + ( i * COLS + col3)));     /*if new o
ne is lower, reasign best */
                        }
                }



            }

/*eliminate all but best*/

            /* calculation of DF error */

            dftemp = (float) sqrt( ((double)dfbest) );
            dftemp = dftemp + ((int) DFDIMERR);
            dferror = dftemp*dftemp;

            /*************************/


            for (i=0;i<filelength;i++) {

                if (      (*(input + ( i * COLS + col4))) == groupticker
```

```
          /*same group*/
                    &&   (*(input + ( i * COLS + col1)))  != NEGONE
      /*within subgroup*/
                    &&   (*(input + ( i * COLS + col3)))  >  (dfbest + dferror)    )   {
                 /*greater then df best*/

                      *(input + (i * COLS + col1)) = NEGONE;
                      groupnumber = groupnumber - 1;

                }
             }

        }


        if (groupnumber > 1)    {       /*i.e. after DF and P best, still more than one, j
udge by V*/

/*is there more than one member in the group left?*/
/*then judge by df */

             firstv = NO;


/*find best df and eliminatring rest as go*/
/*NOTE for V, BEST is HIGHEST*/

             for (i=0;i<filelength;i++) {

                 if (     (*(input + ( i * COLS + col4))) == groupticker
   /*find first member of group and sub group*/
                    &&   (*(input + ( i * COLS + col1)))  != NEGONE
   /*sub-group */
                    &&    firstv == NO                                    ) {



                    crambest = (*(input + (i * COLS + col1)));      /*assign best to first
 in group and sub group*/
                    best = i;
                    firstv = YES;

                 }


                 else if (       firstv == YES                                          /*find
 other members of this group */
                           &&   (*(input + ( i * COLS + col4))) == groupticker
                           &&   (*(input + ( i * COLS + col1)))  != NEGONE  ) {


                       if ( crambest  <  (*(input + ( i * COLS + col1)))     )     {
   /*compare within group*/

                          crambest = (*(input + ( i * COLS + col1)));     /*if new one i
s lower, reasign dfbest */
                          *(input + (best * COLS + col1)) = NEGONE;           /*reassign
 previous best DF as not*/
                          best = i;
                       }

                       if ( crambest  > (*(input + ( i * COLS + col1)))     )      {    /
*if comparison is higher*/
                                *(input + (i * COLS + col1)) = NEGONE;
                       }

                          /* NOTE: currently, if equal, both are kept */
                          /* NEED TO DO SOMETHING ABOUT THIS - WARN USER */
                 }
             }


        }


        groupticker = groupticker + 1;
```

```
    }



    /* Part Three -- OUTPUT the best in each group to file -- print output to file PREDICT
*****/

    sprintf(fullname, "%s.txt", outfile);
    /** OPEN predict FILE for writing **/

    output = fopen(fullname, "w");
    if (output == NULL)      {
        printf("\nFailure to Create Predict File.\n");
        exit(2);
    }


    fprintf(output, "Shevek Predictions\n");
    fprintf(output, "Pos1\tPos2\tCrm V\t-log(P)\tDF\n");

    for (i=0;i<filelength;i++) {


        /**** for debugging*/
        printf("\n%d\t%d\t%d\t%.5f\t%.5f\t%.1f",
                (int) (*(input + (i * COLS + 0))),
                (int) (*(input + (i * COLS + 1))),
                (int) (*(input + (i * COLS + 2))),
                (float) (*(input + (i * COLS + 5))),
                (float) (*(input + (i * COLS + 7))),
                (float) (*(input + (i * COLS + 6)))  );
        /***** end of debugging screen */


        if (  (*(input + (i * COLS + col1)))  != NEGONE)    {

                fprintf(output, "%d\t%d\t%.5f\t%.5f\t%.1f\n",
                    (int) (*(input + (i * COLS + 0))),
                    (int) (*(input + (i * COLS + 1))),
                    (float) (*(input + (i * COLS + 5))),
                    (float) (*(input + (i * COLS + 7))),
                    (float) (*(input + (i * COLS + 6)))  );

                printcounter = printcounter + 1;
        }
    }

    fclose(output);

    if (printcounter > group )  {           /*i.e. there was a redundancy/degeneracy */
                                            /*should only happend of DFs are equal */
        message(6);                         /*WARN USER to check data */

    }

}




void pathfinder(int filepos, int group, double *datafile, int filemax)  {

/*X = value of first position; Y= value of second position, filemax = total number of item
s in PVscan file*/


/*COLS is global offset given file shape  */
```

```
/* Need to put data somewhere */
/* Need to mark old data -- eliminator2 function */


    int xup;
    int xdown;
    int yup;
    int ydown;

    xup = filepos;
    xdown = filepos;
    yup = filepos;
    ydown = filepos;


    (*(datafile + (filepos * COLS + col4))) = group;


    /* Four loops for four 'directions */

    /* Loop One */

    while (xup < (filemax-1))   {

        /* if X finds a match seeking up in either position 0 or position 1*/

        if (
                (    (*(datafile + (filepos * COLS + 0))) == (*(datafile + ( (xup+1) * CO
LS + 0)))
                  || (*(datafile + (filepos * COLS + 0))) == (*(datafile + ( (xup+1) * CO
LS + 1)))
         )
             &&  (  (*(datafile + ( (xup+1) * COLS + col4))) >= 0
             )
           )    {

            pathfinder( (xup+1), group, datafile, filemax);

        }

        xup +=1;

    }


    /*Loop Two */

    while (xdown > 0)   {

        /* if X finds a match seeking down in either position 0 or position 1*/


        if (
                (    (*(datafile + (filepos * COLS + 0))) == (*(datafile + ( (xdown-1) *
COLS + 0)))
                  || (*(datafile + (filepos * COLS + 0))) == (*(datafile + ( (xdown-1) *
COLS + 1)))
         )
             &&  (  (*(datafile + ( (xdown-1) * COLS + col4))) >= 0
             )
           )    {


            pathfinder( (xdown-1), group, datafile, filemax);

        }
```

```
        xdown -=1;
    }

    /* Loop three */
    while (yup < (filemax-1))   {
        /* if y finds a match seeking up in either position 0 or position 1*/
        if (
                (    (*(datafile + (filepos * COLS + 1))) == (*(datafile + ( (yup+1) * CO
LS + 0)))
                  || (*(datafile + (filepos * COLS + 1))) == (*(datafile + ( (yup+1) * CO
LS + 1)))
         )
             &&  (  (*(datafile + ( (yup+1) * COLS + col4))) >= 0
             )
           )    {

            pathfinder( (yup+1), group, datafile, filemax);
        }
        yup +=1;
    }

    /*Loop four */

    while (ydown > 0)   {
        /* if y finds a match seeking down in either position 0 or position 1*/

        if (
                (    (*(datafile + (filepos * COLS + 1))) == (*(datafile + ( (ydown-1) *
COLS + 0)))
                  || (*(datafile + (filepos * COLS + 1))) == (*(datafile + ( (ydown-1) *
COLS + 1)))
         )
             &&  (  (*(datafile + ( (ydown-1) * COLS + col4))) >= 0
             )
           )    {

            pathfinder( (ydown-1), group, datafile, filemax);
        }
        ydown -=1;
    }

}
```

```
/*******************************************************/
/********       Written By Phillip S. Pang      ********/
/*******************************************************/

/*******************************************************/
/******** MD/PhD Candidate, Columbia University ********/
/******** College of Physicians and Surgeons    ********/
/******** Dept. Of Biochemistry and Biophysics  ********/
/*******************************************************/

/*******************************************************/
/********    phillip.pang@stanfordalumni.org    ********/
/*******************************************************/


/*******************************************************/
/********       STATEMENT OF COPYRIGHT          ********/

/*          Copyright 2001 by The Trustees of          */
/*          Columbia University in the City of         */
/*          New York. ALL RIGHTS RESERVED;             */

/*******************************************************/




#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

#include "nrutilp.h"
#include "shevek.h"
#include "definitions.h"


int FILELENGTH;     /*file length of alldata file*/
                    /*determined in distribution analyzer function*/



/*************************************************************/
/************   SCREENER function       **********************/
/*************************************************************/


void screener(int dfilelength, float *VThresh, float *PThresh)
{
    void    distribution_analyzer(int filelength, float *VThresh, float *PThresh);
    void    apply_thresholds(int filelength);
    void    message(int number);



    message(1);     /*print to screen scoring message*/

    distribution_analyzer(dfilelength,VThresh,PThresh);   /*caclulate IQR and thus V&P Thr
esholds */



    printf("\n************************************************");

    printf("\n\nSUGGESTED Lower -Log(P) threhold: %f", *PThresh);
    printf("\n\nSUGGESTED Lower V threshold: %f", *VThresh);

    printf("\n\n************************************************");


    message(2);     /*print to screen threshold warning message*/


    apply_thresholds(FILELENGTH);               /*filelength is of alldata.txt file*/
                                                /*returns filelength of predict.txt file*/
```

```
    message(3);     /*print to screen prediction message*/

}


/*************************************************************/
/************   DISTRIBUTION_ANALYZER function       **********/
/*************************************************************/

void distribution_analyzer(int filelength,float *VThresh, float *PThresh)
{

    float Viqr, Piqr;
    int **input;
    char    filename[50];
    FILE * inputfile = NULL;
    int found_file = 0;

    int **imatrix(long nrl, long nrh, long ncl, long nch);

    void exitprogram();

    int itemp1,itemp2,itemp3,itemp4;

    int i;

    int x;

    int j,k;
    int temp;
    int sumV = 0, sumP = 0;

    float Vmedianloc, Pmedianloc;
    float qlocV, qlocP;
    float firstqV,thirdqV,firstqP,thirdqP;

    extern int FILELENGTH;

/***** OPEN FILE ******/


    while (found_file == 0) {                       /*Queries for input file*/
        printf("\nENTER distribution file name (distribution.txt):");
        scanf("%s", filename);
        inputfile = fopen(filename, "r");
        if (inputfile == NULL)   {
            printf("\nFAILURE TO OPEN OR LOCATE FILE!!!\n");
            found_file = 0;
        }
        else
            found_file = 1;
    }

/**** READ FILE ********/



    printf("\nAllocating Memory for distribution file.\n");

    /** determine file size **/

    filelength = 0;

    x = fgetc(inputfile);
    while (x != EOF) {

        if (x == '\n')  filelength += 1;
        x = fgetc(inputfile);
    }


    /*Allocate memory */
```

```
    input = imatrix(0,filelength,1,4);

    if (input == NULL) {
        printf("\nDynamic Memory Allocation FAILURE!!\n");
        exitprogram();
        exit(1);
    }
    else
        printf("Allocation Completed.\n");



    /*** INPUT FILE INTO MEMORY ***/

    fseek(inputfile, 0, SEEK_SET);  /*start reading from begining of file*/
    printf("Attempting to Read File. Please Wait.\n");


    for(i=0;i < filelength;i++) {

        fscanf(inputfile, "%d %d %d %d", &itemp1,&itemp2,&itemp3,&itemp4);

        input[i][1] = itemp1;  /*V spread */
        input[i][2] = itemp2;   /*V counts */
        input[i][3] = itemp3;   /*-log(P) spread */
        input[i][4] = itemp4;   /*-log(P) counts */

    }

    printf("\n\nDistribution File Read.\n");

    fclose(inputfile);


    /************************************/



/********** Analyze Distributions ******/


    for (j=0; j< filelength; j++) {

        if (input[j][2] != -1)  sumV = sumV + input[j][2];  /*sum total of V counts*/
        if (input[j][4] != -1)  sumP = sumP + input[j][4];  /*sum total of P counts*/

    }


    FILELENGTH = sumV;

    /*printf("\nV sum = %d",sumV); */
    /*printf("\n-log(P) sum = %d\n",sumP);*/


    /* Find medianloc location*/

    Vmedianloc = ( ((float)sumV) + 1) / 2;
    Pmedianloc = ( ((float)sumP) + 1) / 2;

    /*printf("\nV medianloc = %f",Vmedianloc); */
    /*printf("\n-log(P) medianloc = %f\n",Pmedianloc);*/


    /*Find quartile location*/

    qlocV = ( ((float) ((int) Vmedianloc) + 1)) /2;
    qlocP = ( ((float) ((int) Pmedianloc) + 1)) /2;


    /*printf("\nV QL = %f",qlocV); */
    /*printf("\n-log(P) QL = %f\n",qlocP);*/
```

```
    /* find V quartiles */

    temp = 0;
    k =0;

    while (temp < qlocV) {
        temp = temp + input[k][2];
        k = k+1;
    }

    firstqV = (float) input[k][1];      /*first quartile location*/


    temp = 0;
    k = 0;

    while (temp < 3*qlocV) {    /*third quartile location*/
        temp = temp + input[k][2];
        k = k+1;
    }

    thirdqV = (float) input[k][1];

    Viqr = (thirdqV - firstqV)/1000;    /*because distribution is 1000 wide*/


    /*printf("\nV 1Q 3Q = %f\t%f\n",firstqV,thirdqV); */
    /*printf("\nV distance = %f\n",Viqr);*/

    /* find P quartiles */

    temp = 0;
    k =0;

    while (temp < qlocP) {
        temp = temp + input[k][4];
        k = k+1;
    }

    firstqP = (float) input[k][3];      /*first quartile location*/


    temp = 0;
    k = 0;

    while (temp < 3*qlocP) {
        temp = temp + input[k][4];
        k = k+1;
    }

    thirdqP = (float) input[k][3];      /*third quartile location*/

    Piqr = (thirdqP - firstqP) / (100); /*because distribution is 100 wide*/

    /*printf("\n-log(P) 1Q-value 3Q-Value = %f\t%f\n",firstqP,thirdqP);*/

/*Adjust thirdq*/

    thirdqV = thirdqV/1000;
    thirdqP = thirdqP/(100);


    /*printf("\nV 50 boundary (adj) = %f",thirdqV); */
    /*printf("\n-log(P) 50% boundary(adj) = %f\n",thirdqP);*/


    printf("\n-log(P) IQR = %f\n",Piqr);
    printf("-log(P) 3Q-value = %f\n",thirdqP);

    printf("V IQR = %f\n",Viqr);
    printf("V 3Q-value = %f\n",thirdqV);
```

```
    /************************/
    /*Calculate Thresholds*/

    *VThresh = (float) (thirdqV + (VMULT * Viqr));
    *PThresh = (float) (thirdqP + (PMULT * Piqr));

    if (*VThresh < (float) VMINIMUM)
            *VThresh = (float) VMINIMUM;

    if (*PThresh < (float) PMINIMUM)
            *PThresh = (float) PMINIMUM;


    /************************/

}
```

```
/*******************************************************/
/********       Written By Phillip S. Pang      ********/
/*******************************************************/

/*******************************************************/
/******** MD/PhD Candidate, Columbia University ********/
/******** College of Physicians and Surgeons    ********/
/******** Dept. Of Biochemistry and Biophysics  ********/
/*******************************************************/

/*******************************************************/
/********    phillip.pang@stanfordalumni.org    ********/
/*******************************************************/


/*******************************************************/
/********       STATEMENT OF COPYRIGHT          ********/

/*          Copyright 2001 by The Trustees of          */
/*          Columbia University in the City of         */
/*          New York. ALL RIGHTS RESERVED;             */

/*******************************************************/




#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <math.h>

#include "nrutilp.h"
#include "shevek.h"
#include "definitions.h"

int pfilelength = 0;
int OFFSET;
int NUM_ROW;

/*********************************************************/
/***** global structures for putative misalignment data */
/*********************************************************/

struct position {
    int one;
    int two;
};


struct pair {
    char a;
    char b;
};



struct sequence {
    int name;
    struct position pos;/*position*/
    struct pair cur;    /*current*/
    struct pair sug1;
    struct pair sug2;
    struct pair sug3;
};

struct sequence seq[100000];


/*********************************************************/
/*********************************************************/
/*********************************************************/
```

```
/*************************************************************/
/*************************************************************/
/************   MISALIGN_IDENTIFER function ******************/
/*************************************************************/
/*************************************************************/

/* input is pointer to sequence alignment, offset = OFFSET, numrows = NUM_ROWS, and pfilel
ength is
length of predict.txt file */


void misalign_identifier(char *input, int offset, int numrows)
{
    void    misalignment_manager(char *input,int offset,int numrows);
    void    message(int number);


    extern int OFFSET;
    extern int NUM_ROW;

    OFFSET = offset;
    NUM_ROW = numrows;

    message(5);                 /*misalignment INTRODUCTORY TEXT */

    if (numrows > 100000)   {                   /** SEE structure seq declaration **/

        printf("\n\n\n\a\a\a\a\a\aPOSSIBLE MEMORY OVER-RUN in Misalignment Algorithm\n");
        printf(". . .... output is suspect.\n\n\n");
        printf("CONTACT THE PROGRAMER: phillip.pang@stanfordalumni.org");

    }


    misalignment_manager(input,offset,numrows);
    /*Manages multiple matrix files*/


}


/*************************************************************/
/************   MISALIGNMENT_MANAGER function     ************/
/*************************************************************/


void    misalignment_manager(char *input,int offset,int numrows)
{


    int **AR_analysis(char *input, int offset, int numrows, int column1, int column2, int
*num_seq);
    int find_unique_seq(int *inp, int *out, int ns);
    double  *openfile4();

    int **chi_matrix;
    int column1, column2;
    int i,j,tally;

    double  *input4;        /*Predictions Input*/
    int ns = -1;
    int *num_seq;           /*num_seq goes from 0 to num therefore total is actually +1*/
    int num_unique;     /*number of unique sequences*/

    int *inp;               /*1D array of "names"/sequence numbers */
    int *out;               /*array of unique sequences*/ /*see num_unique*/



    char    outfile[40] = "Misalign";           /*output file for data*/
    char    fullname[100];                      /*output file for data*/
```

```
    extern struct sequence seq[100000];

    int printflag = NO;

    FILE * output = NULL;                          /*output file initiation*/
    num_seq = &ns;

    /** open and read file of prediction data **/

    input4 = openfile4();           /*OPENS THEN READS FILE*/
                                    /*returns pointer to predictions data*/



    /** initialize output file **/
    sprintf(fullname, "%s.txt", outfile);
    output = fopen(fullname, "w");

    if (output == NULL)      {
        printf("\nFailure to Create Misalign File.\n");
        exit(2);
    }



    /*** FOR EVERY PAIR IN PREDICT FILE, DO THE FOLLOWING ***/

    for (i=0;i<pfilelength;i++) {

        /* printf("get this far??"); */
        column1 = (int) (*(input4 + (i * PCOLS + 0)));
        column2 = (int) (*(input4 + (i * PCOLS + 1)));
        /* printf("past first read"); */

        /* Calc. AR values, and identify putative misaligned sequences */
        /* assigns them to structure seq, with seq_num being total number of such sequence
s */

        chi_matrix = AR_analysis(input, offset, numrows, column1, column2,num_seq);  /*PRO
BLEM*/

    }



    /*NOW, WITH THIS LIST of putative misaligned sequences, determine which appear frequen
tly*/

    /*printf("past allocation of all seq");*/

    /*** first, convert structure of names to a 1D integer array ***/

    inp = ivector(0,(long) ns);  /*allocate memory; max necessary is num_seq*/
    out = ivector(0,(long) ns);  /*allocate memory; max is num_seq*/   /*IDENTITY OF UNIQU
E SEQUENCES*/

    for (i=0;i<=ns;i++) *(inp+i) = seq[i].name;



    /*** second, now find all the unique sequences in that array***/
    num_unique = find_unique_seq(inp, out, ns);


    if (num_unique == ns) {         /*both start @ ZERO!!! */
            printf("\nNO SEQUENCES IDENTIFYABLE AS MISALIGNED\n.");
            fprintf(output,"NO SEQUENCES IDENTIFYABLE AS MISALIGNED\n.");
        }

    /*** third, now tally number of times each sequence occurs, output as called for ***/


    else {

        fprintf(output, "SEQUENCES IDENTIFIED AS POSSIBLY MISALIGNED:\n");
```

```
        fprintf(output, "(note: sequence numbering starts at 1)\n");
        fprintf(output, "(note: position numbering starts at 0)\n");
        fprintf(output, "(The character '#' denotes NO SUGGESTION POSSIBLE)\n");

        for(i = 0; i <= num_unique; i++)    {

            tally = 0;
            for (j= 0; j<= ns; j++) {
                if (out[i] == seq[j].name) tally +=1;
            }


            /* for each unique sequence, if occurs MIN times IS misaligned */
            if (tally >= MINIMUMSEQ)    {


                printflag = YES;

                printf("\n\n\nSequence number %d is possibly misaligned\n", out[i]);
                fprintf(output, "\n\n*********************************************");
                fprintf(output, "\nSequence number %d is possibly misaligned:\n", out[i]);



                for (j= 0; j<= ns; j++) {
                    if (out[i] == seq[j].name)      {

                        fprintf(output, "\n\tRegion/Positions:");
                        fprintf(output, "%d-%d\t\t", seq[j].pos.one, seq[j].pos.two);
                        fprintf(output, "Units/Characters:");
                        fprintf(output, "%c-%c\n\n", seq[j].cur.a, seq[j].cur.b);
                        fprintf(output, "\tSuggested Alternatives (in order):\n");
                        fprintf(output, "\t%c-%c\n", seq[j].sug1.a, seq[j].sug1.b);
                        fprintf(output, "\t%c-%c\n", seq[j].sug2.a, seq[j].sug2.b);
                        fprintf(output, "\t%c-%c\n", seq[j].sug3.a, seq[j].sug3.b);
                    }
                }

            }
        }
    }


    if (printflag == NO) printf("\n\n\nNO SEQUENCES IDENTIFIED");
    free_imatrix(chi_matrix, 0, POSITIONS, 0, POSITIONS);
    fclose(output);

}



/*************************************************************/
/************   AR_ANALYSIS function     *********************/
/*************************************************************/


int **AR_analysis(char *input, int offset, int numrows, int column1, int column2, int *num
_seq)
{

    int     find_unique_elements2(char *input, int numrows, int offset, char *aa, int *gap
, int var);
    void    crosstab2(char *input, int **chi_matrix, char *aa1, char *aa2, int var1, int v
ar2, int count1, int count2);
    void    arcalc(int **nn, int ni, int nj, float *chisq, float *df, double *prob, float
*cramrv, float *ccc, float **ar);

    void    gap_rectifer1(int **chi_matrix,int gap1, int *cnt1, int *cnt2, char *u1);
    void    gap_rectifer2(int **chi_matrix,int gap2, int *cnt1, int *cnt2, char *u2);

    int cell_finder(char *input,int offset, int numrows, int num_start, char *unique1, cha
r *unique2, int count1, int count2,
                int column1, int column2,int **chi_matx2,float **ar_matrix);

    char    outfile[30] = "ARtables";
    char    fullname[40];
```

```
    char    uni1[POSITIONS];                    /*array of unique aa in a position*/
    char    uni2[POSITIONS];                    /*array of unique aa in a 2nd pos */
    char    *unique1;
    char    *unique2;

    int count1;                             /*number of unique aa in a position*/
    int count2;                             /*number of unique aa in a 2nd pos */

    int temp_numseq;

    int i,j,k;

    float   chi, d, cram, cc;
    double  pr;
    float   *chisq, *df, *cramrv, *ccc;
    double  *prob;

    int **chi_matrix;                       /*CROSSTABULATION matrix of correlation frequency
*/
    int **chi_matx2;                        /*cross tabulation matrix with origin=ORIGIN */
                                            /*This ORIGIN shift is necessary for arcalc functi
on*/
                                            /*can and should be replaced by use of ?vector fun
ctions*/

    int gap1, gap2;                     /*number location of gap (-) in uni arrays*/


    float **ar_matrix;                  /****FOR AR tables!!!! *****/

    FILE * output = NULL;

    unique1 = uni1;
    unique2 = uni2;
    chisq = &chi;
    df = &d;
    prob = &pr;
    cramrv = &cram;
    ccc = &cc;

    chi_matrix = imatrix(0, POSITIONS, 0, POSITIONS);     /*allocates memory for an int ma
trix */

    ar_matrix = matrix(1,POSITIONS+1,1,POSITIONS+1);                                /***FOR AR
****/

    /*Open File for output, entitled tabout.txt*/

    sprintf(fullname, "%s%dx%d.txt", outfile, column1, column2);

    output = fopen(fullname, "w");

    if (output == NULL)      {
        printf("\nFailure to Create File.\n");
        exit(2);
    }


    count1 = find_unique_elements2(input, numrows, offset, unique1, &gap1, column1);
    count2 = find_unique_elements2(input, numrows, offset, unique2, &gap2, column2);


    /*printf("\nFUE reached for postion %d",column1);*/

    crosstab2(input, chi_matrix, unique1, unique2, column1, column2, count1, count2);

    gap_rectifer1(chi_matrix,gap1, &count1, &count2, unique1);
    gap_rectifer2(chi_matrix,gap2, &count1, &count2, unique2);

    chi_matx2 = subimatrix(chi_matrix,0,POSITIONS,0,POSITIONS,ORIGIN,ORIGIN);

    arcalc(chi_matx2, count1+ORIGIN, count2+ORIGIN, chisq, df, prob, cramrv, ccc, ar_matri
x);
```

```
    temp_numseq = *num_seq;

    *num_seq = cell_finder(input,offset,numrows,temp_numseq, unique1, unique2, count1, cou
nt2,
                column1, column2,chi_matx2,ar_matrix);


    /*************************/
    /*** start of display ****/
    /*************************/

    printf("\n\n\n\nActual Frequency Table for %d x %d\n",column1,column2);

    for (i = 0; i <= count2; i++)
            printf("\t%c",unique2[i]);

    printf("\n");

    for (j = 0; j <= count1; j++)   {
            printf("%c",unique1[j]);

        for (k = 0; k <= count2; k++)   {
            printf("\t%d", chi_matrix[j][k]);
        }

        printf("\n");
    }

    /** printf("\nchi:%f df:%f prob:%.15f cram:%f cc:%f\n", chi, d, pr, cram, cc); **/

    /*************************/
    /*** END Of DISPLAY  ****/
    /*************************/

    /*****************************/
    /*** Start of file output ***/
    /*****************************/

    fprintf(output,"Actual Frequency Table:");
    fprintf(output, "%d x %d\n", column1,column2);


    for (i = 0; i <= (count2); i++)
            fprintf(output, "\t%c",unique2[i]);

    fprintf(output, "\n");

    for (j = 0; j <= (count1); j++) {
            fprintf(output, "%c",unique1[j]);

        for (k = 0; k <= count2; k++)   {
            fprintf(output, "\t%d", chi_matrix[j][k]);
        }

        fprintf(output, "\n");
    }

    /** fprintf(output, "\nchi:%f df:%f prob:%.15f cram:%f cc:%f\n\n\n", chi, d, pr, cram,
 cc); **/


    /***output for AR***/

    fprintf(output,"\n\nAdjusted Residual (AR) Table:\n");
    for (j = 1; j <= (count1 + 1); j++) {
        for (k = 1; k <= (count2 + 1); k++) {
            fprintf(output, "\t%.5f\t", ar_matrix[j][k]);
        }

        fprintf(output, "\n");
    }


    /***END OF OUTPUT FOR AR****/

    fclose(output);
```

```
    /*** END of file output***/


    free_subimatrix(chi_matx2, 1, POSITIONS+ORIGIN, 1, POSITIONS+ORIGIN);

    return chi_matrix;    /*return pointer for sole purpose of freeing memory allocation*/

}


/*************************************************************/
/************   FIND_UNIQUE_ELEMENTS2 function     ************/
/*************************************************************/


int find_unique_elements2(char *input, int numrows, int offset, char *aa, int *gap, int va
r)
{


    int NUM_ROW = numrows;

    int i;
    int j;
    int k;
    int OFFSET = offset;
    int r;
    int exit;
    int counter = 0;        /* # of unique amino acids */


    for (k = 0; k < POSITIONS; k++)  {        /*clear buffer*/
        aa[k] = 0;
    }


    *gap = -1;              /* start off = -1; signifies no gap in position, in any sequen
ce*/

    aa[0] = *(input + var);     /*the first amino acid is always unique*/


    for (i = 1; i < NUM_ROW; i++) {         /*now compare REST of aa=NR-1 :therefore < */
        j = counter;
        exit = NO;

        while (exit == NO) {
            if (*(input + (i * OFFSET + var)) != aa[j]) {

                j = j - 1;
                if (j < 0)      /* No Match, therefore unique */ {
                    counter = counter + 1;
                    aa[counter] = *(input + (i * OFFSET + var));
                    exit = YES;
                } else /* No match yet, check previous elements */
                    exit = NO;
            } else /*MATCHED, so exit*/
                exit = YES;
        }
    }


    for (r = 0; r<=counter; r++) {
        if (aa[r] == '-')           {
            *gap = r;
            return counter;
        }
    }

    return counter;
}
```

```
/*************************************************************/
/************   GAP_RECTIFIER1 function     ******************/
/*************************************************************/


void gap_rectifer1(int **chi_matrix,int gap1, int *cnt1, int *cnt2, char *u1)


/* removes gap (-) from chi_matrix   */
/* NOTE: any zero columns and rows   */
/* produced by gap-as-value removal, */
/* removed in chisq funct.           */
/* Two gap removers b/c of iterative nature of program */

{

    int i,j;

    if (gap1 != -1)  {


        for (i = (gap1+1); i <= *cnt1 ; i++ )   {

                u1[i-1] = u1[i];                              /*removes gap label*/

            for (j = 0; j <= *cnt2; j++)            {
                chi_matrix[i-1][j] = chi_matrix[i][j];      /*removes gap values*/
            }
        }

        *cnt1 = *cnt1 - 1;
    }


}


/*************************************************************/
/************   GAP_RECTIFIER2 function     ******************/
/*************************************************************/


void gap_rectifer2(int **chi_matrix,int gap2, int *cnt1, int *cnt2, char *u2)


{

    int k,l;

    if (gap2 != -1) {

    for (k = (gap2+1); k <= *cnt2 ; k++ )   {

                u2[k-1] = u2[k];                              /*removes gap label*/

            for (l = 0; l <= *cnt1; l++)            {
                chi_matrix[l][k-1] = chi_matrix[l][k];      /*removes gap values*/
            }
        }

        *cnt2 = *cnt2 - 1;
    }

}



/*************************************************************/
/************   OPENFILE4 function     ***********************/
/*************************************************************/

double  *openfile4()
{

    double  *read_input4(FILE *ifile);
```

```
    double  *input4;
    char    filename[50];

    FILE * inputfile4 = NULL;

    int found_file = 0;

    while (found_file == 0) {                          /*Queries for input file*/

        printf("\nENTER predictions file name (predict.txt):");

        scanf("%s", filename);
        inputfile4 = fopen(filename, "r");
        if (inputfile4 == NULL)  {
            printf("\nFAILURE TO OPEN OR LOCATE FILE!!!\n");
            found_file = 0;
        }
        else
            found_file = 1;
    }


    input4 = read_input4(inputfile4);
    fclose(inputfile4);

    return input4;

}


/*************************************************************/
/************   READ_INPUT4 function     *********************/
/*************************************************************/


double  *read_input4(FILE *ifile)
{
    double *dvector(long nl, long nh);

    void exitprogram();

    double  *input4;

    double inputtemp;

    int i;

    char x;

    extern int pfilelength;


    /*** Memory allocation: determining file size ***/

    printf("\nAllocating Memory for predict.txt file.\n");


    x = fgetc(ifile);
    while (x != EOF) {

        if (x == '\n')  pfilelength += 1;
        x = fgetc(ifile);
    }

    pfilelength = pfilelength - 2;  /*see output from threshold.c*/


    input4 = dvector(0,pfilelength*PCOLS);

    if (input4 == NULL) {
        printf("\nDynamic Memory Allocation FAILURE!!\n");
        exitprogram();
        exit(1);
    }
    else
```

```
        printf("Allocation Completed.\n");
        printf("Allocated memory for %d predictions\n",pfilelength);


    /*** INPUT FILE INTO MEMORY ***/

    fseek(ifile, 0, SEEK_SET);  /*start reading from begining of file*/
    printf("\n\nAttempting to Read Predictions File. Please Wait.\n");

    i = 0; /*start at begining of memory*/
    input4[0] = 0;
    inputtemp = 0;

    fscanf(ifile, "Shevek Predictions\nPos1\tPos2\tCrm V\t-log(P)\tDF\n");  /*scan through
text at top*/

    while (i < pfilelength*PCOLS) {

        fscanf(ifile, "%lf", &inputtemp);
        input4[i] = inputtemp;

        i = i+1;
    }


    printf("\n\n Predictions File Read.\n");

    return input4;
}



/*************************************************************/
/************   CELL_FINDER function     *********************/
/*************************************************************/


int cell_finder(char *input, int offset, int numrows, int num_start, char *unique1, char *
unique2, int count1, int count2,
                int column1, int column2,int **chi_matx2,float **ar_matrix) {

/*this function returns num_seq --> so that next time it is called, begins there*/
/*number_start begins at zero, but continues upward*/
/* unique is array from 0 to count */
/* chi_matx2 and ar_matrix go from = to count+1 */
/* column strarts at 0 */

    void sorter(float *A, int left, int right);
    int sequence_finder(char *input, int offset, int numrows, int start, char char1, char
char2, int column1, int column2);

    int i;
    int j;
    int temp,temp2,sizetemp;
    int k;
    char charac1;
    char charac2;

    int start;          /*so that if many seq have same character pair, don't return same
sequence*/

    int num_seq;        /*counting of number of sequences entered into array*/

    float *arvector;

    float best1;
    float best2;
    float best3;

    struct pair s1;
    struct pair s2;
    struct pair s3;

    extern struct sequence seq[100000];
```

```
   num_seq = num_start;

/*********************************/
/* main cycle for each AR table */
/*********************************/


   /*****************************************************************/
   /* First determine AR values related to misaligned sequences */
   /*****************************************************************/

   for (i = 1; i <= (count1+ORIGIN); i++)  {
       for (j = 1; j <= (count2+ORIGIN); j++)  {

               /*between 0 and neg threshold*/
           if (   ((ar_matrix[i][j] < 0.0) && (ar_matrix[i][j] >= (ARTHRESH*(-1))) )
               || ((ar_matrix[i][j] > 0.0) && (ar_matrix[i][j] <= ARTHRESH)        )
   ) {

               /*is there is a sequence that corrresponds to that cell?*/
               if (chi_matx2[i][j] > 0)    {

                   temp = chi_matx2[i][j];
                   charac1 = unique1[i-1];
                   charac2 = unique2[j-1];
                   start = 0;

                   for (k = 1; k <= temp; k++) {

                       start = sequence_finder(input,offset,numrows,start,charac1,charac2
,column1,column2);

                       if (start == -1) {
                           printf("ERROR ERROR: LOGIC PATHWAY PROBLEM.");
                           exit(5);
                       }


                       num_seq += 1;

                       seq[num_seq].name = start;
                       seq[num_seq].pos.one = column1;
                       seq[num_seq].pos.two = column2;
                       seq[num_seq].cur.a = charac1;
                       seq[num_seq].cur.b = charac2;

                   }

               }

           }

       }
   }



   /*****************************************/
   /* Second, determine suggested alternatives */
   /*****************************************/



   /*begin by allocating memory for one dimentional array*/
   sizetemp = (count1+ORIGIN)*(count2+ORIGIN);
   arvector = vector(0, sizetemp-1);
   temp2 = 0;

   /*convert two dimentional array into 1 dimentional array*/

   for (i = 1; i <= (count1+ORIGIN); i++)  {
       for (j = 1; j <= (count2+ORIGIN); j++)  {

           arvector[temp2] = ar_matrix[i][j];
```

```
        temp2 += 1;

    }
}

/*sort array in increasing order*/

sorter(arvector,0,sizetemp-1);


/*assign best AR scores to best1...3*/
/*if less then threshold, then not a best score*/

if (arvector[sizetemp-1] > ARTHRESH)    best1 = arvector[sizetemp-1];
else best1 = (float) NEGONE;

if (arvector[sizetemp-2] > ARTHRESH)    best2 = arvector[sizetemp-2];
else best2 = (float) NEGONE;

if (arvector[sizetemp-3] > ARTHRESH)    best3 = arvector[sizetemp-3];
else best3 = (float) NEGONE;


/*Find characters associated with those scores*/

for (i = 1; i <= (count1+ORIGIN); i++)  {
    for (j = 1; j <= (count2+ORIGIN); j++)  {

        if (ar_matrix[i][j] == best1)   {

                charac1 = unique1[i-1];
                charac2 = unique2[j-1];
                s1.a = charac1;
                s1.b = charac2;

        }

        else if (ar_matrix[i][j] == best2)  {

                charac1 = unique1[i-1];
                charac2 = unique2[j-1];
                s2.a = charac1;
                s2.b = charac2;

        }

        else if (ar_matrix[i][j] == best3)  {

                charac1 = unique1[i-1];
                charac2 = unique2[j-1];
                s3.a = charac1;
                s3.b = charac2;
        }

    }
}

/* IF as above, best scores were assigned NEGONE, then suggested chacter is '#' */

if (best1 == NEGONE) {
        s1.a = NOCHAR;
        s1.b = NOCHAR;
}

if (best2 == NEGONE) {
        s2.a = NOCHAR;
        s2.b = NOCHAR;
}

if (best3 == NEGONE) {
        s3.a = NOCHAR;
        s3.b = NOCHAR;
}


/** NOW, assign these suggestions to every sequence from this AR table ***/
```

```
    /** obviously, if no identifed sequences, then don't nothing will be assigned**/

    for (k=(num_start+1); k<=num_seq; k++)    {

        seq[k].sug1.a =  s1.a;
        seq[k].sug1.b =  s1.b;
        seq[k].sug2.a =  s2.a;
        seq[k].sug2.b =  s2.b;
        seq[k].sug3.a =  s3.a;
        seq[k].sug3.b =  s3.b;

    }


    return num_seq;

}


/*************************************************************/
/************   SEQUENCE_FINDER function     *****************/
/*************************************************************/

int sequence_finder(char *input, int offset, int numrows, int start, char char1, char char
2, int column1, int column2) {

    int i;

    int OFFSET = offset;
    int NUM_ROW = numrows;

    for (i = start; i < NUM_ROW; i++)    {
        if (  (  (*(input + (i * OFFSET + column1))) == char1) && (  (*(input + (i * OFFSE
T + column2))) == char2) )
            return (i+1);
    }

    return -1;  /*THIS PATHWAY SHOULD NEVER OCCUR*/
}



/*************************************************************/
/************   FIND_UNIQUE_SEQ function   *******************/
/*************************************************************/

int find_unique_seq(int *inp, int *out, int num_seq)
{

    int i;
    int j;
    int k;

    int exit;
    int counter = 0;        /* # of unique sequences */


    for (k = 0; k <= num_seq; k++)            /*clear buffer*/
        out[k] = 0;                             /* need this??*/


    out[0] = *(inp + 0);        /*the first sequence is always unique*/


    for (i = 1; i <= num_seq; i++) {            /*now compare REST */
        j = counter;
        exit = NO;

        while (exit == NO) {
            if ( *(inp + i) != out[j]) {

                j = j - 1;
```

```
                if (j < 0)      /* No Match, therefore unique */ {
                    counter = counter + 1;
                    out[counter] = *(inp + i);
                    exit = YES;
                } else /* No match yet, check previous elements */
                    exit = NO;
            } else /*MATCHED, so exit*/
                exit = YES;
        }
    }

    return counter;
}

/************************************************/
/********FUNCTION: Crosstab2 **********************/
/************************************************/

/* Description: using two lists of unique amino acids, found in position var1, var2 */
/* compare all aa in a position to the list [0...p or q]                             */
/* when a match is found in position var, add 1 to matrix position p                 */
/* when a match is found in position var2, add 1 to matrix position q                */
/* Thus, chi_matrix[p][q] is a tally of frequency of all possible combinations       */
/* of amino acids found in var1 with var 2,                                          */

/** THIS FUNCTION IS REPEATED because it calls two externals, that are limited to another
file**/


void    crosstab2(char *input, int **chi_matrix, char *aa1, char *aa2, int var1, int var2,
 int count1, int count2)
{
    extern int  OFFSET;
    extern int  NUM_ROW;

    void exitprogram();

    int i, j, k, n, m, p, q;
    int found_p = 0;
    int found_q = 0;

    for (j = 0; j <= POSITIONS; j++) {              /*initialize all values to zero */
        for (k = 0; k <= POSITIONS; k++)   {
            chi_matrix[j][k] = 0;
        }
    }

    for (i = 0; i < NUM_ROW; i++)    {

        n = 0;                      /*n are unique aa in 1*/
        /*input compares to n*/
        m = 0;                      /*m are unique aa in 2*/
        /*input compares to m*/
        found_p = NO;               /*when a match is found, signal*/
        found_q = NO;               /*when a match is found, signal*/


        while (found_p == NO)   {
            if ( n > count1 )   {
                printf("ERROR: overflow of unique characters set 1");
                printf("\n n = %d; i = %d",n,i);
                printf("\n %s = aa1 after overflow", aa1);
                printf("\n%c = input",(*input+(i*OFFSET+var1)));
                exitprogram();
                exit(4);
            }
            if (  *(input + (i * OFFSET + var1))  == aa1[n] )   {
                p = n;
                found_p = YES;
            }
            else {
                n += 1;
                found_p = NO;
            }
        }
```

```
        while (found_q == NO)   {
            if (m > count2 )     {
                printf("ERROR: overflow of unique amino acids set 2");
                exitprogram();
                exit(3);
            }
            if (  *(input + (i * OFFSET + var2))   == aa2[m] )   {
                q = m;
                found_q = YES;
            }
            else {
                m += 1;
                found_q = NO;
            }
        }

        chi_matrix[p][q] += 1;

    }
}
```

```
/*********************************************/
/*********************************************/
/*********************************************/

/* standlib functions as defined from ANSI C */

/*********************************************/
/*********************************************/
/*********************************************/



/*************************************************************/
/************   SORTER function   ****************************/
/*************************************************************/

void sorter(float *A, int left, int right) {

    int i, last;
    void swapper (float *A, int i, int j);

    if (left >= right)
        return;
    swapper(A, left, (left+right)/2);
    last = left;

    for (i=left+1; i <=right; i++)
        if ( (*(A+i)) < (*(A+left))  )    swapper(A, ++last, i);


    swapper(A,left,last);
    sorter(A,left, last-1);
    sorter(A,last+1,right);
}


/*************************************************************/
/************   SWAPPER function   ***************************/
/*************************************************************/



void swapper(float *A, int i, int j)    {

    float temp;

    temp = (*(A+i));
    (*(A+i)) = (*(A+j));
    (*(A+j)) = temp;
```

```
}
```

```
/*********************************************************/
/********       Written By Phillip S. Pang       ********/
/*********************************************************/

/*********************************************************/
/******** MD/PhD Candidate, Columbia University ********/
/******** College of Physicians and Surgeons    ********/
/******** Dept. Of Biochemistry and Biophysics  ********/
/*********************************************************/

/*********************************************************/
/********    phillip.pang@stanfordalumni.org    ********/
/*********************************************************/


/*********************************************************/
/********       STATEMENT OF COPYRIGHT           ********/

/*          Copyright 2001 by The Trustees of          */
/*          Columbia University in the City of         */
/*          New York. ALL RIGHTS RESERVED;             */

/*********************************************************/



/*****************************************************************/
/*****************************************************************/
/*****************************************************************/
/*                                                               */
/*  The algorithms found within this file may be derivatives     */
/*  of source code obtained from the book:                       */
/*  "Numerical Recipes in C: The Art of Scientific Computing"    */
/*  published by Cambridge University Press.                     */
/*                                                               */
/*****************************************************************/
/*****************************************************************/
/*****************************************************************/

#include <math.h>

#include "nrutilp.h"
#include "definitions.h"


/*************************************************************/
/************   ARCALC function     **************************/
/*************************************************************/


void arcalc(int **nn, int ni, int nj, float *chisq, float *df, double *prob, float *cramrv
, float *ccc, float **ar)

/*Given a two-dimensional contingency table in the form of an integer array nn[1..ni][1..n
j],
this routine returns the chi-square chisq, the number of degrees of freedom df, the signi_
cance
level prob (small values indicating a signi_cant association), and two measures of associa
tion,
Cramer's V (cramrv) and the contingency coe_cient C (ccc).  */

/*Since you pass the address, and fill the value, then of course, the address you passed,
will have a value in it!*/
/*this is how multiple values are returned! */

{
float gammq(float a, float x);
int nnj,nni,j,i;
float sum=0.0,expctd,*sumi,*sumj,temp;

double temp2;           /****for AR calc***/

sumi=vector(1,ni);
sumj=vector(1,nj);
nni=ni;                                 /*Number of rows*/
nnj=nj;                                 /*and columns.*/
```

```
for (i=1;i<=ni;i++) {                    /*Get the row totals.*/
    sumi[i]=0.0;
        for (j=1;j<=nj;j++) {
            sumi[i] += nn[i][j];
            sum += nn[i][j];
        }
    if (sumi[i] == 0.0) --nni;           /*Eliminate any zero rows by reducing the num*/
}


for (j=1;j<=nj;j++) {                    /*Get the column totals.*/
    sumj[j]=0.0;
    for (i=1;i<=ni;i++) sumj[j] += nn[i][j];
    if (sumj[j] == 0.0) --nnj;           /*Eliminate any zero columns.*/
}

 /* *df=nni*nnj-nni-nnj+1;  */           /*Corrected number of degrees of freedom.*/
*chisq=0.0;

for (i=1;i<=ni;i++) {                    /*Do the chi-square sum.*/
    for (j=1;j<=nj;j++) {
    expctd=sumj[j]*sumi[i]/sum;
    temp=nn[i][j]-expctd;
    /*  *chisq += temp*temp/(expctd+TINY);  */  /*Here TINY guarantees that any*/ /*elim.
Div. By zero */

    temp2 = expctd * (1 - sumi[i]/sum) * (1 - sumj[j]/sum);

    ar[i][j] = (float) ( (temp) / sqrt(temp2)  );     /***AR calc***/

        /*** printf("\n%f %f\n", temp2, ar[i][j]);  ***/

    }                              /* eliminated row or column will*/
}

                              /*not contribute to the sum.*/

    /* *prob=gammq(0.5*(*df),0.5*(*chisq));     */  /*Chi-square probability function.*/
    /* minij = nni < nnj ? nni-1 : nnj-1;       */
    /* *cramrv=sqrt(*chisq/(sum*minij));        */
    /* *ccc=sqrt(*chisq/(*chisq+sum));          */

free_vector(sumj,1,nj);

free_vector(sumi,1,ni);
}
```

```
/*********************************************************/
/********       Written By Phillip S. Pang       ********/
/*********************************************************/

/*********************************************************/
/******** MD/PhD Candidate, Columbia University ********/
/******** College of Physicians and Surgeons    ********/
/******** Dept. Of Biochemistry and Biophysics  ********/
/*********************************************************/

/*********************************************************/
/********    phillip.pang@stanfordalumni.org    ********/
/*********************************************************/


/*********************************************************/
/********       STATEMENT OF COPYRIGHT          ********/

/*          Copyright 2001 by The Trustees of          */
/*          Columbia University in the City of         */
/*          New York. ALL RIGHTS RESERVED;             */

/*********************************************************/



/************************************************************************/
/************************************************************************/
/************************************************************************/
/*                                                                      */
/* THIS CODE WAS TRANSLATED, OPTIMIZED, and WRITTEN by RAUL Shakirov    */
/* from FORTRAN TO C, As a work-for-hire by request of Phillip S. Pang */
/*                                                                      */
/* Contact information for Raul: http://www.imach.uran.ru/rns/          */
/*                                                                      */
/*                                                                      */
/* It has been further modified by Phillip S. Pang                      */
/*                                                                      */
/************************************************************************/
/************************************************************************/
/************************************************************************/



/* ----------------------------------------------------------------------
    Header file for fastexp2 function.
---------------------------------------------------------------------- */

extern void fastexp2ini (void);
extern void fastexp2inc (void);
extern double fastexp2  (double value);
```

```
/*********************************************************/
/********       Written By Phillip S. Pang      ********/
/*********************************************************/

/*********************************************************/
/******** MD/PhD Candidate, Columbia University ********/
/******** College of Physicians and Surgeons    ********/
/******** Dept. Of Biochemistry and Biophysics  ********/
/*********************************************************/

/*********************************************************/
/********    phillip.pang@stanfordalumni.org    ********/
/*********************************************************/


/*********************************************************/
/********       STATEMENT OF COPYRIGHT          ********/

/*          Copyright 2001 by The Trustees of          */
/*          Columbia University in the City of         */
/*          New York. ALL RIGHTS RESERVED;             */

/*********************************************************/



/************************************************************************/
/************************************************************************/
/************************************************************************/
/*                                                                      */
/* THIS CODE WAS TRANSLATED, OPTIMIZED, and WRITTEN by RAUL Shakirov    */
/* from FORTRAN TO C, As a work-for-hire by request of Phillip S. Pang  */
/*                                                                      */
/* Contact information for Raul: http://www.imach.uran.ru/rns/          */
/*                                                                      */
/*                                                                      */
/* It has been further modified by Phillip S. Pang                      */
/*                                                                      */
/************************************************************************/
/************************************************************************/
/************************************************************************/


/* ----------------------------------------------------------------------
    Common declarations for rcont2 function.
---------------------------------------------------------------------- */

#include <assert.h>     /* Include standard assert function */
#include <limits.h>     /* Header file for implementation specific values */
#include <stdlib.h>     /* Header file for standard functions */
#include "fastexp2.h"   /* Header file for fastexp2 function */
#include "support.h"    /* Header file for support functions */


long double pangrand(void);

/* ----------------------------------------------------------------------
    Restrictons:
    maxtot      Maximal total sum for the contingency table
---------------------------------------------------------------------- */

#define maxtot 100000   /* Maximal total sum */

/* ----------------------------------------------------------------------
    Debug pointers and memory allocation functions.

    PLEASE, READ THIS CAREFULLY BECAUSE THE FOLLOWING
    IS NOT COMMON PRACTICE FOR C/C++ PROGRAMMING!

    All rcont2 functions, except for rcont2_ access
    arrays thereby debug pointers instead of common
    C pointers.

    Debug pointer acts as common C pointer and also
    "knows" size of assigned array to perform bound
    check when arrays elements are accessed.
```

```
    If program attempt to access element outside
    array bounds, debug pointer calls prterr42(),

    Check of bounds requires C++ compiler with templates.
    To disable check of bounds either define macro NDEBUG
    or NCHECKPTR or use C compiler. If so, debug pointers
    will be implemented as common C pointers with no bound
    check (please, look for ANSI C implementation below).

    EXPTR(T) p;      - Declare debug pointer p to array
                       of type T.

    EXPTRTO(T,m,n)   - Construct debug pointer to existing
                       array m of n elements of type T.
                       Array may be represented thereby its
                       name, C pointer or debug pointer.
                       If the latter case, EXPTRTO macro
                       performs bound check.

    EXPTRCHECK(p,n)  - Construct common C pointer for
                       debug pointer p and check if
                       elements 0..n-1 are accessible.

    EXPTRNEW(T,n)    - Allocate array for n elements of
                       type T and return its debug pointer.

    EXPTRDELETE(p)   - Deallocate array, allocated by
                       EXPTRNEW.
---------------------------------------------------------------------- */

#ifdef __cplusplus

/*
    Implementation for C++ with templates.
    Performs bound check if no NDEBUG or NCHECKPTR macro is defined
    Requires files exarray.h and exarray.c
*/

#include "exarray.h"    /* Header file for memory allocation functions */

#else /*__cplusplus*/

/*
    Implementation for ANSI C. No bound check.
    Uses custom sysalloc/sysfree functions from support package.
    Note, that EXPTRNEW allocated one extra item.
    This eliminates non-predictable consequences of common
    programming bug - indexing of non-existing element just
    after the last element. Bound check catches this error
*/

#define EXPTR(T)        T*
#define EXPTRTO(T,m,n)  (m)
#define EXPTRCHECK(p,n) (p)
#define EXPTRNEW(T,n)   ((T*) sysalloc ((n + 1), sizeof(T)))
#define EXPTRDELETE(p)  (sysfree (p))

#endif/*__cplusplus*/

/* ----------------------------------------------------------------------
    Constant values

    MAXTABLEROWS - Maximum number of rows in the contingency table.
    MAXTABLECOLS - Maximum number of cols in the contingency table.
    MAXTABLESUM  - Maximum sum of row/col in the contingency table.
---------------------------------------------------------------------- */

#define MAXTABLEROWS (SHRT_MAX / 2)  /* Suggest << SQRT (INT_MAX)       */
#define MAXTABLECOLS (SHRT_MAX / 2)  /* Suggest << SQRT (INT_MAX)       */
#define MAXTABLESUM  (SHRT_MAX / 2)  /* Suggest << SQRT (INT_MAX)       */

/* ----------------------------------------------------------------------
    Parameter passing conventions:

    1. Parameters type * represent variables, passed by
       rererence thereby C pointers.
```

```
        Input variables have const modifiers to prevent their
        accidental modification within function. Output and
        input/output variables have no const modifiers to allow
        their modification within function.

        For example: double *pre
                     const int nrow

        To pass a variable as a reference, use expression &name.

        For example:     &pre
                         &nrow

    2. Some functions accepts input variables as values.
       Appropriate parameter has no * sign and may have,
       or not to have const modifier.

        For example:     int size
                         int nitems

        To pass a value use name of variable or any expression:

        For example:     size
                         nrow + ncol

    3. Parameters markes as EXPTR(type) represent arrays,
       passed by their debug pointers.

        Input arrays have const modifiers to prevent their
        accidental modification within function. Output and
        input/output arrays have no const modifiers to allow
        their modification within function.

        For example:     EXPTR(double) fact
                         const EXPTR (int) irow

        To pass an array, use debug pointer of array or expression
        debug pointer + index.

        For example:     fact
                         irow + 1

        To produce debug pointer, allocate array thereby macro
        EXPTRNEW or apply macro EXPTRTO to existing array.

        For example:     EXPTR (double) fact = EXPTRNEW (double, 400)
                         static int rows [400];
                         EXPTR (int) irow = EXPTRTO (int, rows, 400)

        Arrays, allocated by EXPTRNEW, are to be deallocated
        by means of EXPTRDELETE function.

        For example:     EXPTRDELETE (fact)
------------------------------------------------------------------------ */

/*
    Variables and functions defined in rcont2.c
*/

extern int   rcont2s (int nrow, int ncol,
                      int *nrowt,  int *ncolt,
                      int *matrix);


extern float rcont2f (void);

/*
    Function defined in rcont2_.c

    This function is converted by means of f2c
    and is included for the test purposes - its
    results are compared with result of manually
    converted rcount2 function.
*/
```

```
extern struct { float hop; } tempry_;

extern int  rcont2_ (int *nrow, int *ncol, int *nrowt, int *ncolt,
                     int *jwork, int *matrix, int *key, int *ifault);

/*
    Functions and variables defined in prterr.c
*/

extern void prcerr (int icode, const char *mes);
extern void prcerr1 (void);
extern void prcerr2 (void);
extern void prcerr3 (void);
extern void prcerr4 (void);
extern void prcerr5 (void);
extern void prcerr6 (void);

/* ------------------------------------------------------------------------
    Name:       RAND

    Purpose:    Generate random value in the range [0..1)

    Usage:      RAND

    Result:     The random value.
------------------------------------------------------------------------ */
#define RAND ((long double) pangrand());

/*#define RAND ((float) rand() / (float) RAND_MAX)*/
```

```
/*****************************************************/
/********       Written By Phillip S. Pang      ********/
/*****************************************************/

/*****************************************************/
/******** MD/PhD Candidate, Columbia University ********/
/******** College of Physicians and Surgeons    ********/
/******** Dept. Of Biochemistry and Biophysics  ********/
/*****************************************************/

/*****************************************************/
/********    phillip.pang@stanfordalumni.org    ********/
/*****************************************************/


/*****************************************************/
/********       STATEMENT OF COPYRIGHT          ********/

/*          Copyright 2001 by The Trustees of          */
/*          Columbia University in the City of         */
/*          New York. ALL RIGHTS RESERVED;             */

/*****************************************************/



/*********************************************************************/
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/* THIS CODE WAS TRANSLATED, OPTIMIZED, and WRITTEN by RAUL Shakirov  */
/* from FORTRAN TO C, As a work-for-hire by request of Phillip S. Pang */
/*                                                                   */
/* Contact information for Raul: http://www.imach.uran.ru/rns/       */
/*                                                                   */
/*                                                                   */
/* It has been further modified by Phillip S. Pang                   */
/*                                                                   */
/*********************************************************************/
/*********************************************************************/
/*********************************************************************/



/* ---------------------------------------------------------------------
    Heder file for support functions.
--------------------------------------------------------------------- */

extern void* sysalloc(int nitems, int size);
extern void  sysfree (void *p);
extern void  syschk  (void);

extern void syserr   (int icode, const char *mes);
extern void syserr40 (void);
extern void syserr41 (void);
extern void syserr42 (void);
extern void syserr43 (void);
extern void syserr50 (void);

#define PCOUNT 100              /* Number of profiling counters */
extern pcount [PCOUNT];         /* Profiling counters */

extern void pcinit (void);
extern int  ptrace (int n);
extern void pctype (void);


#define ptrace(n) (++pcount[n])

/* ---------------------------------------------------------------------
    Name:       RAND

    Purpose:    Generate random value in the range [0..1)

    Usage:      RAND
```

```
    Result:     The random value.
------------------------------------------------------------------------ */

/* #define RAND ((float) rand() / (float) RAND_MAX)  */

/* #define RAND PANGRAND(); */
```

```
/*******************************************************/
/********       Written By Phillip S. Pang      ********/
/*******************************************************/

/*******************************************************/
/******** MD/PhD Candidate, Columbia University ********/
/******** College of Physicians and Surgeons    ********/
/******** Dept. Of Biochemistry and Biophysics  ********/
/*******************************************************/

/*******************************************************/
/********    phillip.pang@stanfordalumni.org    ********/
/*******************************************************/


/*******************************************************/
/********       STATEMENT OF COPYRIGHT          ********/

/*          Copyright 2001 by The Trustees of          */
/*          Columbia University in the City of         */
/*          New York. ALL RIGHTS RESERVED;             */

/*******************************************************/



/** Standard Definitions **/

#define NO  0
#define YES 1


 /** ChiSquare and estimating probability routines -- from numerical recipies **/

#define TINY 1.0e-30    /*A small number.*/
#define ITMAX 1000      /* previously 100 */
#define EPS 3.0e-7
#define FPMIN 1.0e-30


/** Cochran Conditions Parameters -- determine how P value is calculated **/

#define PERCENT     0.80
#define EXPECTED    5.0
#define MINIMUM     1.0


/** Parameter limiting the number of different characters possible in a given position **/

#define POSITIONS   40


/** Statistical calculations are not always sensible, given certain data sets **/

#define MEANINGLESS -1     /*If statistical calculations are not possible, value set to -1
*/


/** Used when calling chisquare or arcalc functions -- resets matrix orgin to 1,1 **
/** Eventually, this can be removed by redefining matrix which uses this        **/

#define ORIGIN 1

/** Multiplier for distributions **/

#define PMULT 2
#define VMULT 1

/** USED in threshold.c as definition of number of columns of data output in data files **
/

#define COLS  8         /*NUMBER OF COL IN FILE*/

#define col1  5         /*COLUMN FOR CRAMER'S V SCORE*/
#define col2  7         /*COLUMN FOR - LOG P SCORE*/
```

```
#define col3  6          /*COLUMN FOR DF          */
#define col4  2          /*previous COLUMN for X2, set to GROUP NUMBER*/


/** Parameters used in misalignment functions **/


#define PCOLS  5                /*Number of columns in predict.txt file -- see threshold.c
 */
#define MINIMUMSEQ 2
                            /*Number of times sequence appears to be considered misaligned
*/
#define ARTHRESH 1.0        /*Threshold for AR cellvalues -- distinguish misaligned from n
ot*/
#define NOCHAR '#'          /*Character default if no suggested alternatives are possible
*/


/** Variable as flag -- when wish to indicate that a variable has meaningless or unused va
lue **/

#define NEGONE  -1


/** Parameter for error and min on logP and V **/

#define VERROR 0.05
#define VMINIMUM 0.40

#define PERROR 0.0
#define PMINIMUM 2.0

#define DFDIMERR 1
```

```
/*************************************************/
/********       Written By Phillip S. Pang      ********/
/*************************************************/

/*************************************************/
/******** MD/PhD Candidate, Columbia University ********/
/******** College of Physicians and Surgeons    ********/
/******** Dept. Of Biochemistry and Biophysics  ********/
/*************************************************/

/*************************************************/
/********    phillip.pang@stanfordalumni.org    ********/
/*************************************************/


/*************************************************/
/********       STATEMENT OF COPYRIGHT          ********/

/*          Copyright 2001 by The Trustees of          */
/*          Columbia University in the City of         */
/*          New York. ALL RIGHTS RESERVED;             */

/*************************************************/



/***************************************************************/
/***************************************************************/
/***************************************************************/
/*                                                             */
/*  The algorithms found within this file may be derivatives   */
/*  of copyright source code obtained from the book:           */
/*  "Numerical Recipes in C: The Art of Scientific Computing"  */
/*  published by Cambridge University Press.                   */
/*                                                             */
/***************************************************************/
/***************************************************************/
/***************************************************************/


#ifndef _NR_UTILS_H_
#define _NR_UTILS_H_

static float sqrarg;
#define SQR(a) ((sqrarg=(a)) == 0.0 ? 0.0 : sqrarg*sqrarg)

static double dsqrarg;
#define DSQR(a) ((dsqrarg=(a)) == 0.0 ? 0.0 : dsqrarg*dsqrarg)

static double dmaxarg1,dmaxarg2;
#define DMAX(a,b) (dmaxarg1=(a),dmaxarg2=(b),(dmaxarg1) > (dmaxarg2) ?\
        (dmaxarg1) : (dmaxarg2))

static double dminarg1,dminarg2;
#define DMIN(a,b) (dminarg1=(a),dminarg2=(b),(dminarg1) < (dminarg2) ?\
        (dminarg1) : (dminarg2))

static float maxarg1,maxarg2;
#define FMAX(a,b) (maxarg1=(a),maxarg2=(b),(maxarg1) > (maxarg2) ?\
        (maxarg1) : (maxarg2))

static float minarg1,minarg2;
#define FMIN(a,b) (minarg1=(a),minarg2=(b),(minarg1) < (minarg2) ?\
        (minarg1) : (minarg2))

static long lmaxarg1,lmaxarg2;
#define LMAX(a,b) (lmaxarg1=(a),lmaxarg2=(b),(lmaxarg1) > (lmaxarg2) ?\
        (lmaxarg1) : (lmaxarg2))

static long lminarg1,lminarg2;
#define LMIN(a,b) (lminarg1=(a),lminarg2=(b),(lminarg1) < (lminarg2) ?\
        (lminarg1) : (lminarg2))

static int imaxarg1,imaxarg2;
#define IMAX(a,b) (imaxarg1=(a),imaxarg2=(b),(imaxarg1) > (imaxarg2) ?\
        (imaxarg1) : (imaxarg2))
```

```
static int iminarg1,iminarg2;
#define IMIN(a,b) (iminarg1=(a),iminarg2=(b),(iminarg1) < (iminarg2) ?\
        (iminarg1) : (iminarg2))

#define SIGN(a,b) ((b) >= 0.0 ? fabs(a) : -fabs(a))


void nrerror(char error_text[]);
float *vector(long nl, long nh);
int *ivector(long nl, long nh);
unsigned char *cvector(long nl, long nh);
unsigned long *lvector(long nl, long nh);
double *dvector(long nl, long nh);
float **matrix(long nrl, long nrh, long ncl, long nch);
double **dmatrix(long nrl, long nrh, long ncl, long nch);
int **imatrix(long nrl, long nrh, long ncl, long nch);
int **subimatrix(int **a, long oldrl, long oldrh, long oldcl, long oldch,
    long newrl, long newcl);
float **convert_matrix(float *a, long nrl, long nrh, long ncl, long nch);
float ***f3tensor(long nrl, long nrh, long ncl, long nch, long ndl, long ndh);
void free_vector(float *v, long nl, long nh);
void free_ivector(int *v, long nl, long nh);
void free_cvector(unsigned char *v, long nl, long nh);
void free_lvector(unsigned long *v, long nl, long nh);
void free_dvector(double *v, long nl, long nh);
void free_matrix(float **m, long nrl, long nrh, long ncl, long nch);
void free_dmatrix(double **m, long nrl, long nrh, long ncl, long nch);
void free_imatrix(int **m, long nrl, long nrh, long ncl, long nch);
void free_subimatrix(int **b, long nrl, long nrh, long ncl, long nch);
void free_convert_matrix(float **b, long nrl, long nrh, long ncl, long nch);
void free_f3tensor(float ***t, long nrl, long nrh, long ncl, long nch,
    long ndl, long ndh);


#endif /* _NR_UTILS_H_ */
```

```
/*******************************************************/
/********       Written By Phillip S. Pang      ********/
/*******************************************************/

/*******************************************************/
/******** MD/PhD Candidate, Columbia University ********/
/******** College of Physicians and Surgeons    ********/
/******** Dept. Of Biochemistry and Biophysics  ********/
/*******************************************************/

/*******************************************************/
/********    phillip.pang@stanfordalumni.org    ********/
/*******************************************************/


/*******************************************************/
/********       STATEMENT OF COPYRIGHT          ********/

/*          Copyright 2001 by The Trustees of          */
/*          Columbia University in the City of         */
/*          New York. ALL RIGHTS RESERVED;             */

/*******************************************************/



void    arcalc(int **nn, int ni, int nj, float *chisq, float *df, double *prob, float *cra
mrv, float *ccc, float **ar);
void    apply_thresholds(int filelength);
int     **chi_analysis(char *input, int start, int stop);
int     chisquare(int **nn, int ni, int nj, float *chisq, float *df, double *prob, double
*cramrv, float *ccc);
int     cochrantest(int **chi_matrix, int *rowtot, int *coltot, float *expctd, int numrows
, int numcols);
float   **convert_matrix(float *a, long nrl, long nrh, long ncl, long nch);
void    crosstab(char *input, int **chi_matrix, char *aa1, char *aa2, int var1, int var2,
int count1, int count2);
void    distribution_analyzer(int filelength,float *VThresh, float *PThresh);
void    eliminator(double *input,int filelength);
double  estExact(int *ROWmatrix, double PRECISION, int numrows, int numcols);
double  fastexp2 (double value);
int     find_unique_elements(char *input, char *aa, int *gap, int var);
void    gap_rectifer(int **chi_matrix,int gap1, int gap2,int *cnt1, int *cnt2);
void    intro();
void    main ();
void    matrixconverter(int **chi_matrix, int *matrix, int count1, int count2, int flag);
void    misalign_identifier(char *input, int offset, int numrows);
void    message(int number);
char    *openfile();
double  *openfile2(int pass, int filelength);
void    positionrelater(char *input);
int     prelimscan(double *input, int filelength);
float   rcont2f (void);
int     rcont2s (int nrow, int ncol,int *_nrowt,int *_ncolt,int *_matrix);
void    score_manager(char *input);
void    screener(int dfilelength, float *VThresh, float *PThresh);
double  *dvector(long nl, long nh);
void    free_dvector(double *v, long nl, long nh);
void    exitprogram();
```

****
****
WARNING WARNING WARNING WARNING WARNING WARNING
****
****

The number of groups (as defined by linkage between positions)
is less then the number of predictions, indicating that one
of the predictions is redundant.

This is most likely due to a redundancy in DF values, P values,
and very similar V values.

For absolute accuracy, my best suggestion is to eliminate both
of these predictions.

Alternatively, run the program for higher P;

A final alternative is to call up the frequency table for these
two interactions, and by inspection attempt to determine which
one seems reasonable. (Don't know the best way to do this.)

****

```
**********************************************************************
**********************************************************************
        Thresholds Applied: Preliminary Analysis Complete
**********************************************************************
**********************************************************************

Shevek has used the input thresholds to perform a preliminary screen
of all the possible associations, eliminating all associations
not possessing the threshold criteria.

The output from this preliminary screen can be viewed under file:
'pvscan.txt'

Shevek will not eliminate all intersecting interactions within this
list.

**********************************************************************
```

```
**********************************************************************
**********************************************************************
        Misalignment Identifcation Process
**********************************************************************
**********************************************************************
```

Shevek will now attempt to identify misaligned sequences. Shevek will output a file entitled, 'misalign.txt', which will list sequences identified as possibly misaligned by categorical statistical analysis.

This information can be used to re-align the sequence alignment. After such re-alignment, rerun Shevek with the new alignment for increased prediction accuracy.

Stop such iteration when the number/identity of possibly misaligned sequences no longer changes -- or further re-alignment is not deemed possible.

```
**********************************************************************
```

```
**********************************************************************
**********************************************************************
        PREDICTIONS COMPLETE: SEE 'predict.txt'
**********************************************************************
**********************************************************************

Note: predictions identify alignment positions. In order to identify
how these alignment positions relate to sequence of interest, see
the file 'position.txt.'

**********************************************************************
```

```
**********************************************************************
**********************************************************************
        Scoring and Standardization Accomplished.
**********************************************************************
**********************************************************************

Please view your alldata file using any plotting program.
Suggested Plots: P versus V; and -logP versus V

Shevek will now calculate suggested lower thresholds for V and -logP
Use these thresholds to help choose an appropriate area.

**********************************************************************
```

```
**********************************************************************
**********************************************************************
        Threshold Application Process (Screening)
**********************************************************************
**********************************************************************
```

USER: having reviewed the suggested plots as well as the suggested thresholds caclulated above, Shevek will now request the thresholds that you have decided upon. If you are unsure of what to enter, use as default values the thresholds stated above.

WARNINGS:

(1) Important Notice:

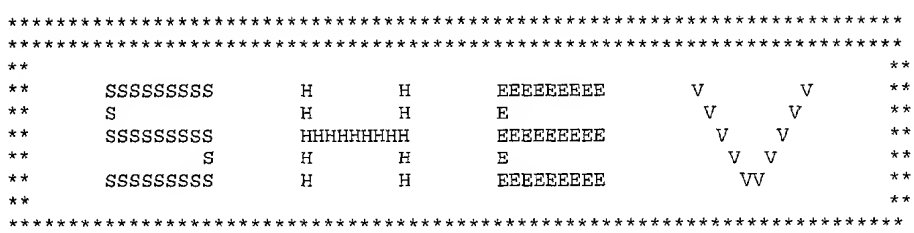
The suggested thresholds represent values that will minimize the possibility of false-positives. Consequently, selection of thresholds lower than the suggested ones may result in false predictions. If no data points are above these thresholds, see warning number (2) and/or: Altering the sequence set by the addition of more sequences and/or increase the diversity of the sequences that are chosen.

(2) Given The Above:

At the begining of the program, an INTEGER PRECISION was entered for the calculation of the P score. If the suggested -log(P) threshold is greater than that entered PRECISION, then Shevek should be rerun with a PRECISION value at least +1 greater than the suggested -log(P) threshold. In other words, if you entered 4 for the PRECISION and the suggested -log(p) threshold is 5, then it is highly advised that Shevek be rerun with a precision equal to or greater than 6.

```
**********************************************************************
```

Welcome to Shevek



In honor of Ursula K. LeGuin

BY

PHILLIP S. PANG

MD-PhD Candidate
Columbia University
phillip.pang@stanfordalumni.org